# Algorithms

# Topic 1 : Why Study Ds And Algorihms

## 1.1 Why Study : Data Structures & Algorithms

- Before we study any subject it is very important to know why you are studying that subject/topic to understand the subject properly. Where are these concepts going to be applied in the real world.

- The most important subjects in computer science
  1. Knowledge of programming
  2. Data Structures and Algorithms

### Some Example Applications

1. Auto Complete on google homepage :- There are many algorithms which are at work behind it.
2. Google Search Engine
3. Search on Google/FB/Amazon.
4. Big Data :- Large volumes of data are to be processed using many data structures and algorithms.
5. Friend Recommendations :- On FB.
6. Uber Cab / OLA Cabs :- Transportation systems.
7. Video Streaming

- Every application makes use of many ds and algorithms.
- DS and Algorithms are the most important subjects in cs.

# TOPIC 2 :- SORTING & SEARCHING : WHY BOTHER WITH THESE SIMPLE TASKS

## VIDEO 2.1 Sorting & Searching Why bothers with these simple tasks.

← Array index

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a: | 1 | 3 | 2 | 7 | 6 | 4 | 8 | 9 |

Unordered array.

A sorted array is one in which the elements are either added in ascending or descending order.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a': | 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 |

← Sorted in ascending order.

$a'_0 \le a'_1 \le a'_2 \le a'_3 \le a'_4 \le a'_5 \le a'_6 \le a'_7$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a'': | 9 | 8 | 7 | 6 | 4 | 3 | 2 | 1 |

← Sorted in descending order.

$a''_0 \geq a''_1 \geq a''_2 \geq a''_3 \geq a''_4 \geq a''_5 \geq a''_6 \geq a''_7$

- Sorting is the process of arranging / rearranging the elements of the array in such a way that the elements are in ascending or descending order.

## SEARCHING

Given array a

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 2 | 7 | 8 | 9 | 4 | 6 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

and a query for example 1 or 10 we need to search for that query if it is present in the array at that location or not.

Why do we need to study / bother about these problems?

- These are applied at many ecommerce sites such as Amazon, Flipkart etc in order to search for a particular product, sort the particular search results in a particular order, etc.
- Even on Facebook friend recommendations are sorted in the order such that the person who has maximum / more chance of being known is put up at the first position and these people are ordered accordingly.

# Topic 3. Insertion Sort

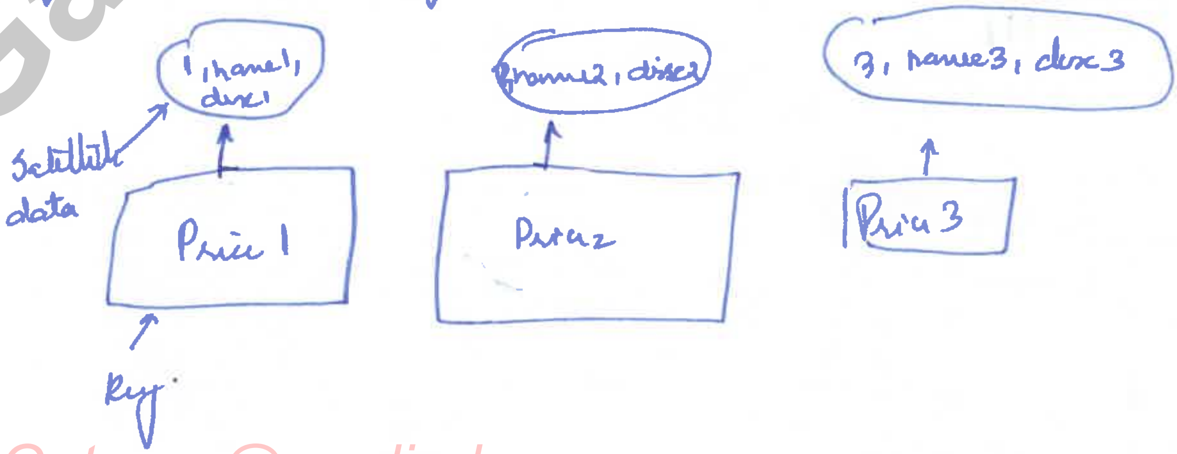
## 3.1 Satellite Data and Key.

Suppose we have data about products on Amazon

| id | name | price | description |
|---|---|---|---|
| | | | |

We search for a particular product and the search results we would like to sort by price because we want to prefer cheaper items in this case the key is the price and the remaining columns are the satellite data.

If we are sorting by price



I/p Array

| | | | | | |
|---|---|---|---|---|---|
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ |

R - Satellite data of every element.

Sorted Array

| | | | | |
|---|---|---|---|---|
| $P_3$ | $P_1$ | $P_4$ | $P_2$ | $P_6$ |

Sorted array based on the key but there may be satellite data available with the key which moves along with it!

- It is called as satellite data, because it moves along with the ~~key~~ similar to how a satellite moves along with the earth/planet.

---

## 3.2. How It Works: Card Sorting

### Insertion Sort

I/p :- a

| 6 | 5 | 3 | 1 | 8 | 7 | 2 | 4 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Unsorted array

① First element is already sorted in the sub array A[1-1]

② Consider the second element it is smaller than the 1st element.

It is stored in a temporary element and swapped with element 1

| 5 | 6 | 3 | 1 | 8 | 2 | 4 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

The sub array A[1-2] is sorted.

(3) The third element is considered, 3. it is smaller than the element at location 2, it is stored in the temp variable K.

K = 3, Second element is moved towards the right
Now it is compared with the 1st element as well it is smaller so the 1st element is moved towards the right and K is placed at location 1

| 3 | 5 | 6 | 1 | 8 | 7 | 2 | 4 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Now after this step the sub array A[1-3] is already sorted and the remaining part A[4-8] is unsorted.

(4) The fourth element is compared with the remaining elements and it gives the following array.

| 1 | 3 | 5 | 6 | 8 | 7 | 2 | 4 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Array A[1-4] is sorted

(5) The fifth element A[5] i.e. 8 it is compared with the 4th element it is greater and not movements are done.

| 1 | 3 | 5 | 6 | 8 | 7 | 2 | 4 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

A[1-5] is sorted.

(6) Now the K = 7 ; 6th element , it is placed before 5th element

A

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 6 | 7 | 8 | 2 | 4 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Now A[1..6] is sorted

(7) K = 2 , seventh element , it is compared and moved down to position

A

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 6 | 7 | 8 | 4 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

A[1-7] is sorted .

(8) K=4 , it is moved down to position 4

A

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8. |

Now we have the entire array is sorted.

– The insertion sort is similar to the card sorting process because a person with cards in his hand will sort it in a similar fashion.

# 3.3 PSEUDO-CODE

## INSERTION-SORT (A)

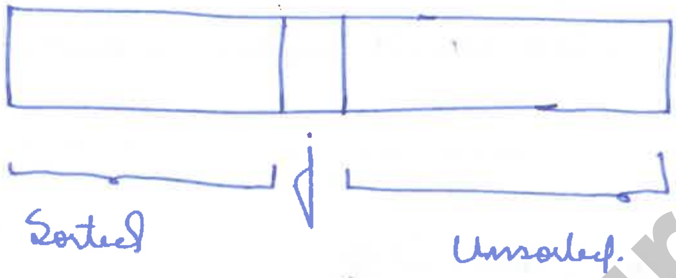
```
1. for j=2 to A.length.
2.     key = A[j]
3.     // Insert A[j] into sorted A[1....j-1]
4.     i = j-1
5.     while i>0 AND A[i]> key.
6.             A[i+1] = A[i]
7.             i = i-1
8.     A [i+1]= key.
```

→ The outer loop ranges from 2 to A.length it iterates from the second position till the end of the array at every step that

- line 2 key is assigned as the jth element.
- line 4 i is initialized as j-1 to compare with the elements preceding element j or the key.
- lines 5 to 7 the key is compared with preceding elements and swapped if it is smaller than the preceding element, this loop is repeated until a larger element is found or the 1st element is reached.
- At line 6 the key is inserted at the correct location.

## 3.4 CORRECTNESS

- How do we prove that this algorithm insertion sort will always work or is correct?




At the end of $j^{th}$ iteration (for a given value of j) the array A[1 . . . j] is sorted, and before the start of the $j^{th}$ iteration the array A[1 . . . j-1] is sorted

As j moves from 2 to n the sorted sub list grows from single element to complete list, this will ensure that insertion sort will correctly sort the elements, this is an intuitive for understanding the correctness of the insertion sort algorithm.

## 3.5. INPLACE SORTING

- Inplace Sorting is a property of a sorting algorithm.
- If the size of the input array is n.

- In Insertion sort we are using additional space for only 3 variables key, i, j and the sorting is taking place in the same input array without using any additional space, in other words the sorting is happening inplace.

- Insertion sort is an inplace sorting algorithm, there may be are other sorting algorithms which are not inplace in nature.

---

## 3.6. Stable Sort

→ Stability is a property of a sorting algorithm.

→ If the ordering of the repeated elements is preserved after the sorting is performed then such a sorting algorithm is stable sorting algorithm.

- Why is stability important?

Let us consider the following products example

Sorted by name.

| Price | name |
|---|---|
| 23 | apple |
| 36 | lenovo |
| 31 | acer |
| 31 | asus |
| 40 | HP |
| 60 | lenovo |

Sort by name (1) →

| Price | Name |
|---|---|
| 31 | acer |
| 23 | apple |
| 31 | asus |
| 40 | HP |
| 36 | lenovo |
| 60 | lenovo |

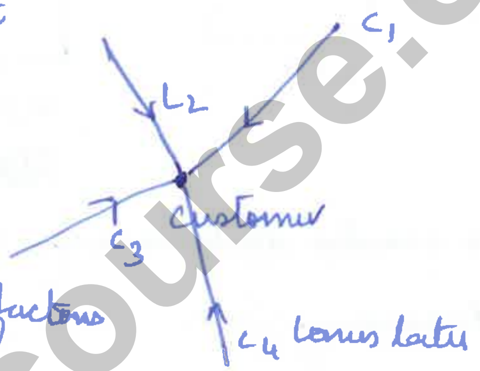
Sort by price (2)

| | |
|---|---|
| 23 | apple |
| 31 | acer |
| 31 | ASUS |
| 36 | lenovo |
| 40 | HP |
| 60 | lenovo |

Here the order is preserved As US also acer is before in this table because of stable sorting.

## 3.7 Online Sorting

- It is also a property of sorting algorithm.
- While studying Insertion sort we have assumed that the complete unsorted array is available for us before hand / before starting the algorithm. But in many real world situations the complete list of array elements may not be available at the beginning.
- An example would be of Uber cab request there may be a set of drivers which are available based on their distance from the customer / rating and other factors



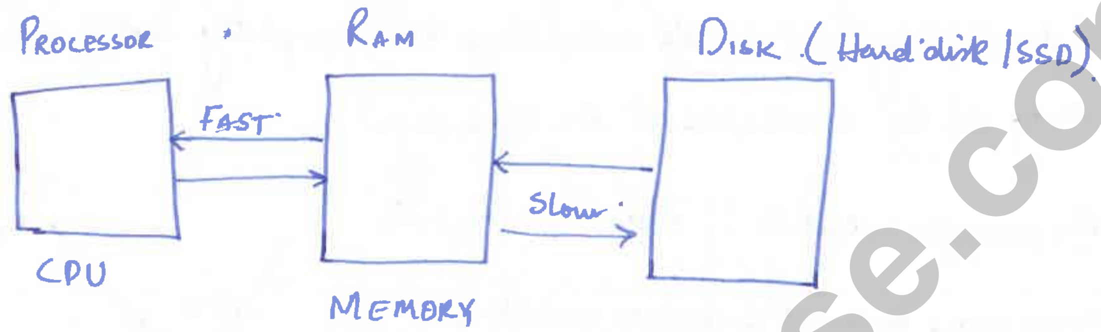

the algorithm will return the best suited driver to the customer, but the list of drivers need not be fixed it can be dynamic and changing at any instant there can be another driver which is more closer than any other driver. then the algorithms must take care of such dynamic changes in the input and work accordingly, such an algorithm is an online algorithm.

- Data can arrive at any point of time for the online algorithm.
- If insertion sort has sorted an array at any time a new elements arrives it can be compared and sorted and added to that already sorted array.
- Insertion sort is an online sorting algorithm.

Topic: Analyzing An Algorithm

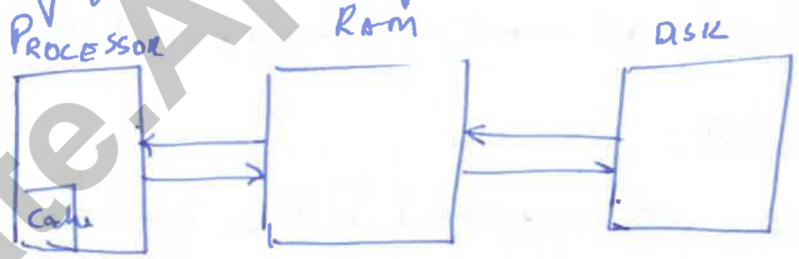
# Video 4.1: Model Of Computation

## Basic Model Of A Computer.

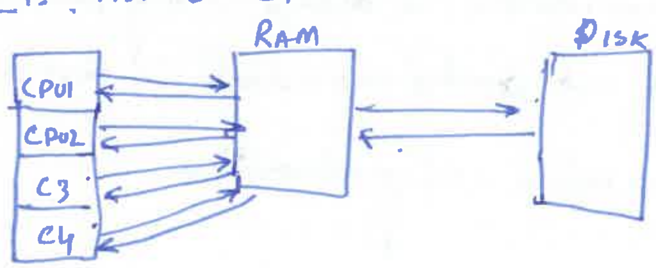



All data for computation sits on the RAM.

- Once the computer shuts down the RAM information / contents are lost / temporary or volatile memory.
- The Disk is the persistant storage. where files and data can be stored permanently.
- Cache :- Most CPU's have an internal memory known as cache which is very fast to access by the CPU than the RAM.




Multi processor Model. Here our CPU will contain multiple processor / cores.

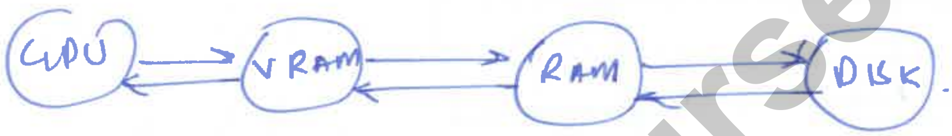


→ Model of Computing for Scientific Computing, Graphics, ML

→ GPU :- Graphics Card.

→ 1000's of processors can access the RAM,

→ Especially for scientific computing, Gaming, Machine learning.




— Our Analysis of algorithms will be based on the simple model which is the first one discussed.

## Video 4.2. Space And Time Analysis Of Insertion Sort – 1

– Let us assume that the length of the array A is n.

→ line 1 of insertion sort (loop check) will execute n times, if it takes $c_1$ µs once, total time $= c_1 \times n$

→ line 2 will execute (n−1) times as the loop executes (n−1) times $= c_2$ µsec $c_2 \times (n-1)$

→ line 4 also (n−1) $= c_3$ µsec for one execution $= (n-1)\times c_3$ µsec.

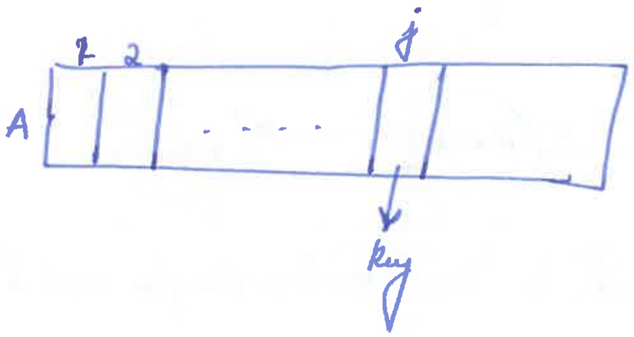
→ line 8 also executes (n−1) times $= c_8$ µsec for one execution $= (n-1)\times c_8$ µsec

Total time $= c_1 \times n + c_2 \times (n-1) + c_3(n+1) + c_8(n-1)$ µsec.

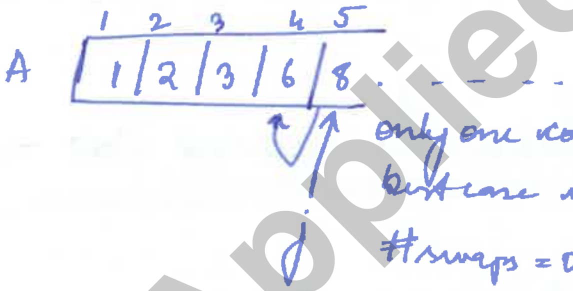
# Video 4.3 Space And Time Analysis Of Insertion Sort - 2

— Lines 5-7 contain the loop which is doing the comparison.

It compares the key with the preceding elements until the correct position for key is determined.



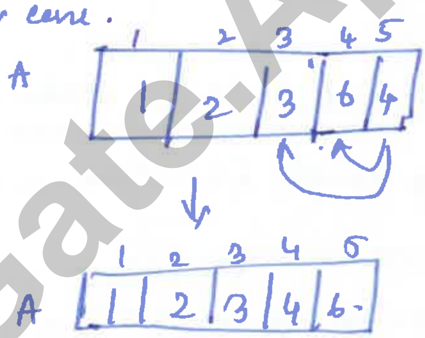

Best case.




only one comparison is to be performed in the best case when $A[j] > A[j-1]$

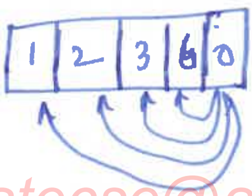
#swaps = 0.

Another case.




Here 2 comparisons are performed and one swap.

Worst case.




# of comparisons = 4 # of swaps = 4.

max # comparisons = $j-1$ # of swaps = $j-1$

– The loop lines 3–7 will run for different values of j.

| | j=2 | j=3 | j=4 | . . . . . | j=n | | |
|---|---|---|---|---|---|---|---|
| Min comparison | 1 | 1 | 1 | | 1 | $\rightarrow (n-1)$ | |
| Max comp | 1 | 2 | 3 | | (n−1) | $\rightarrow 1+2+3 \ldots +(n-1) = \frac{(n-1)\times n}{2}$ | |
| Min # of swaps | 0 | 0 | 0 | | 0 | $\rightarrow 0$ | |
| Max # of swaps | 1 | 2 | 3 | | (n−1) | $\frac{n(n-1)}{2}$ | |

– Now if line 5 takes $C_5$ Msec for one execution it will take $(n-1) \times C_5$ time in best case and in worst case it will take $C_5 \frac{n(n-1)}{2}$.

– Similarly for line 6 and 7 in best case they will execute 0 times and in the worst case they will execute $\frac{n(n-1)}{2}$ times if they take $C_6$ and $C_7$ Msec respectively then time required would be $\frac{n(n-1)}{2} \times C_6$ and $\frac{n(n-1)}{2} C_7$ Msec respectively.

– If we sum up in the best case we will get: $an+b$. where a and b are constants and n is size of the I/p array

– If we sum up in the worst case we will get $a'n^2+b'n+c'$ where $a'$, $b'$ and $c'$ are constants.

→ SPACE COMPLEXITY: – We are just making use of additional 3 variables which is constant in nature.

(16)



Ph: 844-844-0102

# Topic: Big O, Theta, Omega Notation

## Video 6.1 Insertion Sort: Big O-notation

Best case time complexity $= an+b$. a and b are constants

Worst case time complexity $= a'n^2+b'n+c'$ a', b' and c' are constants.

Space complexity = 3 variables

n is the length of the array $n = A.length$.

→ As n increases ~~a~~n + ~~b~~×1 (Best case) also increases

if we ignore the constants

$n+1$ n increases 1 remains constants.

- It can be written as $O(n)$.

→ As n increases the worst case time complexity $a'n^2+b'n+c\times 1$

if we ignore the constants $n^2+n+1$

both $n^2$ and n increase as $n^2$ increases but and n also increases but $n^2$ dominates n and grows faster where as 1 remains constant, therefore we can write it as $O(n^2)$.

Note
- O(n) means as n grows the ~~size~~ time taken grows proportional to n.
similarly $O(n^2)$ means that time taken grows proportional to $n^2$.



Mail: Gatecse@appliedcourse.com

→ The space complexity is 3 variables which is actually a constant does not change as the size of the input changes we can write it as $O(1)$.

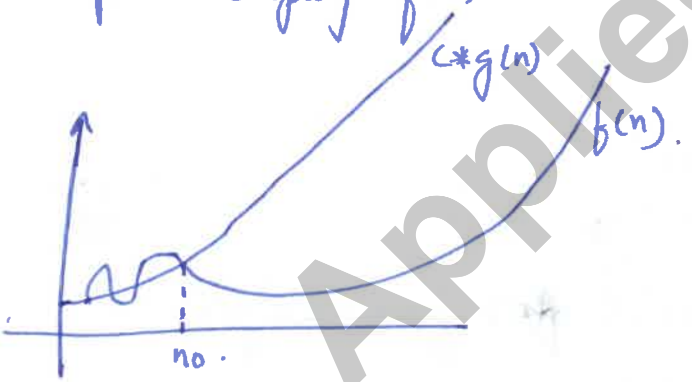
---

# Video 6.2. Notations Big O

We know from the previous video

Time
- Best case $f(n) = an + b*1 \rightarrow O(n)$
- Worst case $f(n) = a'n^2 + b'n + c \rightarrow O(n^2)$

Space Complexity $f(n) = 3 = c \rightarrow O(1)$




$f(n) = O(g(n))$ if and only if there exists $n_0$ & $c$ such that

$$0 \leq f(n) \leq c*g(n)$$
for all $n \geq n_0$.

- If we are able to find the constants $c$ and $n_0$ we can define the big Oh for a given function.

(18)





Let us take the example of. $f(n) = 2n^2 + n + 3$

$a$ $\quad b = 1 \quad c$

lets take $c = 10$ $\quad g(n) = n^2$

$\therefore \quad c \times g(n) = 10n^2$.

$2n^2 + n + 3 <= 10n^2$

if we take $n_0 = 1$, and divide both the sides with $n^2$

i.e.

$2 + \frac{1}{n} + \frac{3}{n^2} <= 10$.

$n = 1 \quad 6 <= 10$

$n = 2 \quad 2 + \frac{1}{2} + \frac{3}{4} <= 10$

$n = 3 \quad 2 + \frac{1}{3} + \frac{3}{9} <= 10$.

$\vdots$

For all values of $n \geq n_0 (1)$ $\quad f(n) <= c\, g(n)$.

$\therefore$ We can write it as $O(n^2)$.

→ In case of $O$ notation $f(n)$ is bound by $g(n)$ and $g(n)$ is the

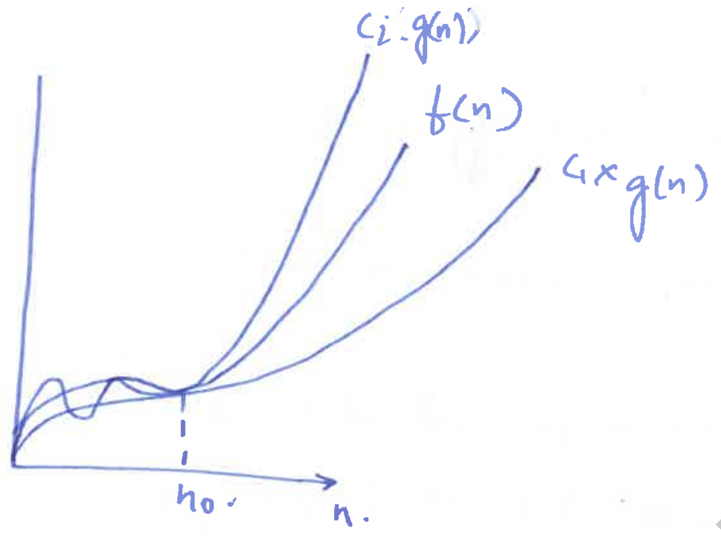
<u>Upper bound</u> for $f(n)$.

# 6.3 Big Omega, Theta





## Big Theta




$f(n) = \Theta(g(n))$ iff there exists $n_0, c_1, c_2$ such that

$0 <= c_1 \times g(n) <= f(n) <= c_2 \times g(n)$ for all $n \geq n_0$.

→ In case of $\Theta$ notation $f(n)$ is tightly bound by $g(n)$.

## Big Omega


$f(n) = \Omega(g(n))$ iff there exists $n_0, c$ such that $0 <= c * g(n) <= f(n)$ for all $n \geq n_0$.

# 6.4 Notations: Small O, Omega, Theta

## Small o

$f(n) = o(g(n))$ iff $\forall c > 0 \quad \exists n_0 > 0$ such that $0 <= f(n) < c \cdot g(n) \ \forall n \geq n_0$.

For example $f(n) = 2n$. we can write it as $O(n)$ but not $o(n)$ because $\forall c > 0$ it is not true that $f(n) < c\,g(n)$

if we take $c = 1$

$2n < n$ does not hold.

We can write $2n = o(n^2)$ $\quad 2n < cn^2 \ \forall c$ for $n \geq n_0$.

$c = 1 \quad n_0 = 20$

$2n < n^2$ holds.

Another alternate definition $f(n) = o(g(n))$ iff $\lim\limits_{n \to \infty} \dfrac{f(n)}{g(n)} = 0$

## Small Omega ($\omega$)

→ $f(n) = \omega(g(n))$ iff $g(n) = o(f(n))$

$2n = o(n^2) \Rightarrow n^2 = \omega(n) \quad \omega(2n) = \omega(n)$.

→ $f(n) = \omega(g(n))$ iff $\forall c>0\ \exists n_0$ such that $0 \le c \cdot g(n) < f(n)\ \forall n \ge n_0$

Alt defn:
→ $f(n) = \omega(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$.

## 6.5 Relationships Between Various Notations

If $f(n) = O(g(n))$ then we can say $f \le g$

$f(n) = \Omega(g(n))$ then $f \ge g$.

$f(n) = \Theta(g(n))$ then $f = g$

$f(n) = o(g(n))$ then $f < g$

$f(n) = \omega(g(m))$ then $f > g$.

→ If $f(n) = \Theta(g(n))$ AND $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$

More intuitively if $f = g$ and $g = h$ then $f = h$ (Transitive relationship).

- The above transitive relationship holds also for $O, \Theta, \Omega, o, \omega$ notation.

22





## Reflexive relation

$f(n) = \Theta(f(n)) \rightarrow f = f$

$f(n) = O(f(n)) \rightarrow f \leq f$

$f(n) = \Omega(f(n)) \rightarrow f \geq f$

$f(n) \neq o(f(n)) \rightarrow f > f$ ×

$f(n) \neq \omega(f(n)) \rightarrow f < f$ ×

## Symmetric relation

$f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$

$f = g$ iff $g = f$

does not hold for $O, \Omega, o, \omega$.

## Transpose symmetric relation

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

$f \leq g$ iff $g \geq f$.

Trichotomy :- If we have 2 numbers a, b, one of the 3 relationships between the numbers always holds either a < b or a > b or a = b.

But in case of O, Θ, Ω, o, ω we can't say for any given two function f(n) and g(n).

ex
$$f(n) = n$$
$$g(n) = n^{1+\sin n}$$ } we can't say for sure as sin n varies in between -1 and 1.

## 6.6 Order Of Common Functions & Real World Applications

| $n$ | $n^2$ | $n \log_{10} n$ | $2^n$ |
|---|---|---|---|
| 1 | 1 | 1.0 | 2 |
| 10 | 100 | $10\times1=10$ | $2^{10}=1024 \simeq 10^3$ |
| 100 | $10^4$ | $100\times2=200$ | $2^{100} = 1.26\times10^{30}$ |
| 1000 | $10^6$ | $1000\times3=3000$ | $2^{1000} = 1.51\times10^{301}$ |
| $10^4$ | $10^8$ | $4\times10^4$ | V.V large. |
| $10^5$ | $10^{10}$ | $5\times10^5$ | V.V.V. large. |

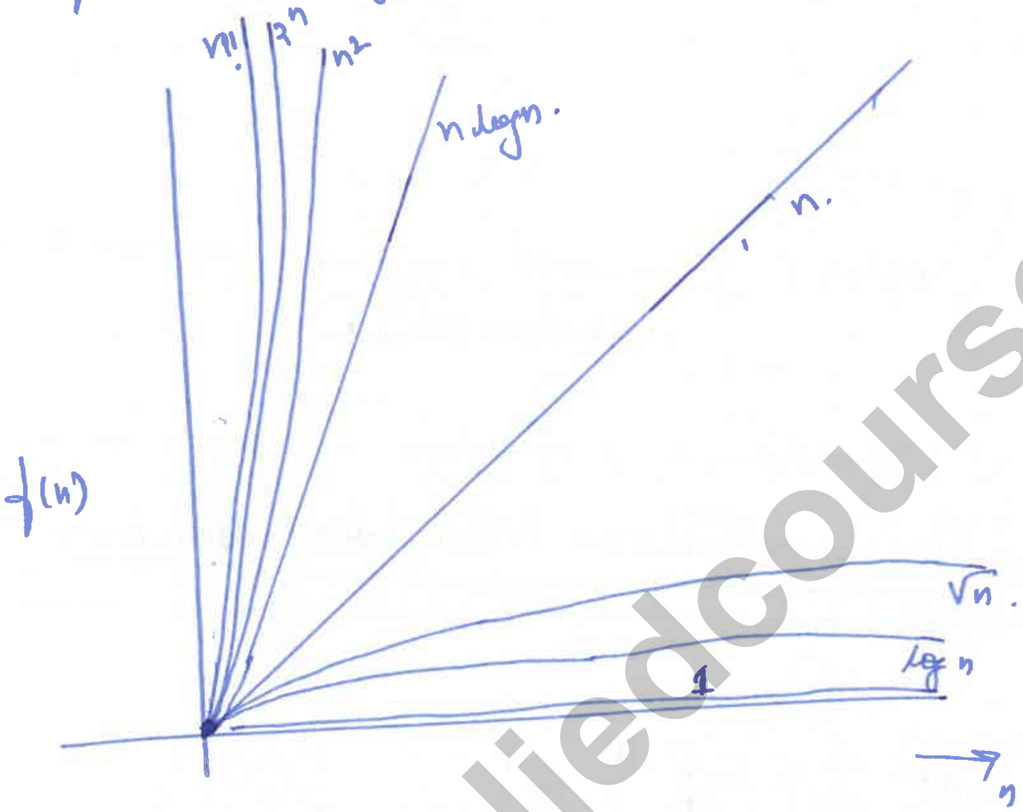
×10 (n), $\times10^2$ ($n^2$), ×20 ($n \log_{10} n$), $\times10^{27}$ ($2^n$)

As n increases these functions also increase with a higher rate

# Order of functions

— $n < n\log n < n^2 < 2^n$

Graph from Wikipedia Page.




→ Mostly in the real world we use O notation

- From the above figure $1 < \log n < \sqrt{n} < n < n\log n < n^2 < 2^n < n!$
- If we know the time complexities and the order of notations we can compare algorithms and decide which one is better.

→

| n | $\log^* n$ |
|---|---|
| $(-\infty, 1]$ | 0 |
| $(1, 2]$ | 1 |
| $(2, 4)$ | 2 |
| $(4, 16)$ | 3 |
| $(16, 65536]$ | 4 |
| $(65536, 2^{65536}]$ | 6 |

How $\log^* n$ is defined

http://en.wikipedia.org/wiki/Big-O-notation#Orders_of_common_functions



| Notation | Name |
| --- | --- |
| $O(1)$ | constant |
| $O(\log(\log n))$ | double logarithmic |
| $O(\log n)$ | logarithmic |
| $O((\log n)^c)$, $c>1$ | polylogarithmic |
| $O(n^c)$, $0<c<1$ | fractional power |
| $O(n)$ | linear |
| $O(n \log^* n)$ | n log-star n |
| $O(n\log n) = O(\log(n!))$ | linearithmic |
| $O(n^2)$ | quadratic |
| $O(n^c)$ | Polynomial |
| $O(c^n)$ | exponential |
| $O(n!)$ | factorial |

26

→ The above table is taken from wikipedia link mentioned
the functions a listed in increasing order of growth.

---

# 6.7. Why Does Asympotic Analysis Matter In Real World

| n | log n | n | n log n | $n^2$ | $2^n$ |
|---|---|---|---|---|---|
| $10^3$ | 9.96 | $10^3$ | 9965.78 | $10^6$ | $1.07 \times 10^{301}$ |
| $10^6$ | 19.93 | $10^6$ | $1.99 \times 10^7$ | $10^{12}$ | — |
| $10^9$ | 29.89 | $10^9$ | $2.99 \times 10^{10}$ | $10^{18}$ | — |

1 hr $= 3.6 \times 10^{12}$ ns

1 day $= 8.64 \times 10^{13}$ ns

1 week $= 6.05 \times 10^{14}$ ns

1 month $= 2.62 \times 10^{15}$ ns

1 year $= 3.15 \times 10^{16}$ ns

Since Big Bang $= 4.32 \times 10^{26}$ ns.

each operation $= 1$ ns $= 10^{-9}$ s

1 billion ns = 1 sec.

The numbers $10^3, 10^6, 10^9$ are not unheard of $10^3$ there are many data sets / operations on this size scale.

- $10^9$ is very common for large giant companies such as Amazon/Google as they billions of queries / visitors on their site every single day.

If we have a logarithmic fn for sorting say $10^9$ times / billion items

$\log n \simeq 30$ ns.

and if we have a $n \log n$ fn it takes $\simeq 3 \times 10^{10}$ ns $\simeq 30$ sec

30 ns Vs 30 sec. It makes a huge difference.

---

## 6-8. Solved Problems: Polynomials

(8) $f(n) = n^2+n+1$

$f(n) = O(g(n))$

$f(n) = \Omega(h(n))$

$f(n) = \Theta(k(n))$.

What are the valid functions for $f(n), g(n); h(n), k(n)$.

Ans we have $f(n) = n^2+n+1$

for $O$ notation $O(g(n))$

let us assume $0 \leq f(n) \leq c\, g(n) \;\forall\, n \geq n_0 \;\exists\, c, n_0$.

we know

$$n^2 <= n^2$$

$$n^2+n+1 \leq n^2+n^2+n^2 \quad \forall n \geq 1$$

$f(n)$ $\qquad$ $c \times g(n)$ say.

$c = 3$ $g(n) = n^2$

Let us take $n^3$ now

$$n^2 <= n^3$$

$$n^2+n+1 <= n^3+n^3+n^3$$

$$f(n) <= 3n^3 \quad \forall n \geq 1$$

$c$ $\qquad$ $g(n)$

$c = 3$ $g(n) = n^3$

Similarly we can take any higher order polynomial in $n$.

If we take $g(n) = n$

$$n^2+n+1 \leq c \cdot n \quad \forall n \geq n_0$$

$n+1+\frac{1}{n} \leq c$. $\qquad$ It does not hold for higher values of $n$

Now let us consider $\Omega$ notation

$f(n) = \Omega(h(n))$ let us take $h = n^2$

$c \cdot h(n) \leq f(n) \; \forall n \geq n_0 \; \exists c.$

$1 \cdot n^2 \leq n^2 + n + 1 \quad \forall n \geq 1$

$c = 1$

$n_0 = 1$

The above inequality holds.

now let us consider $h(n) = n$

$f(n) = n^2 + n + 1$

$c \cdot n \leq n^2 + n + 1 \quad \forall n \geq n_0 \; \exists c, n_0$

$n \leq n^2 + n + 1 \quad c = 1 \quad n \geq 1 \quad n_0 = 1$



Generalization

If we have $f(n)$ is a polynomial

$f(n) = a_0 + a_1 n^1 + a_2 n^2 + a_3 n^3 + \cdots \; a_m n^m.$

we can write

$f(n)\; \cancel{\theta}(g(n))$

$f(n) = \Omega(h(n))$

$f(n) = \Theta(k(n))$

Ph: 844-844-0102

Then for $f(n) = n^2 + n + 1$

$g(n) = n^2, n^3, n^4, n^5 \ldots$ any higher order of $n$.

$h(n) = n^2, h(n) = n \ldots$ (lower orders are accepted here)

$K(n) = n^2$ (Here we have a tight bound and only the highest degree of $f(n)$ is accepted).

for the generalization we can have.

$f(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \ldots + a_m n^m$

1. $f(n) = O(n^m), O(n^{m+1}), O(n^{m+2}) \ldots O(n^{m+k})$ $k \geq 0$
2. $f(n) = \Omega(n^m), \Omega(n^{m-1}), \Omega(n^{m-2}), \Omega(n^{m-k})$ $k \geq 0$
3. $f(n) = \theta(n^m)$

---

## 6.9 Solved Problem : $n > n_0$ case

Q) $f(n) = \begin{cases} n^2 & n \leq 100 \\ n & n > 100 \end{cases}$  $g(n) = \begin{cases} n & n < 1000 \\ n^3 & n \geq 1000 \end{cases}$

(a) $f(n) = O(g(n))$  (b) $g(n) = O(f(n))$



Mail: Gatecse@appliedcourse.com

Ans.

Defn $f(n) \le c \cdot g(n) \;\; \forall \, n \ge n_0 \;\; \exists \, c, n_0 \rightarrow f(n) = O(g$

$n_0 = 1000 \quad \forall \, n \ge n_0 \quad f(n) = n$

$1000. \quad g(n) = n^3.$

$n \le c \, n^3 \quad \forall \, n \ge 1000$

$f(n) = O(g(n))$

---

## 6.10 SOLVED PROBLEM: GATE 2001

Q) let $f(n) = n^2 \log n$ and $g(n) = (\log n)^{10}$ be two positive functions of n. Which of the following statements is correct?

(a) $f(n) = O(g(n))$ and $g(n) \ne O(f(n))$

(b) $g(n) = O(f(n))$ and $f(n) \ne O(g(n))$

(c) $f(n) \ne O(g(n))$ and $g(n) \ne O(f(n))$

(d) $f(n) = O(g(n))$ and $g(n) = O(f(n))$

# 6.11 SOLVED PROBLEM GATE 2011



Q) $f_1(n) = 2^n$, $f_2(n) = n^{3/2}$ $f_3(n) = n\log_2 n$ $f_4 = n^{\log_2 n}$

What is the increasing order of complexity?

Order of func?

$f_3 < f_2$ $f_3 = n\log n < n^{3/2}$

$\cancel{n}\log n < \cancel{n}\, n^{1/2}$

$\log n < n^{1/2}$ is true

$\therefore f_3 < f_2$

Now comparing $f_4$ and $f_2$

$n^{3/2} < n^{\log_2 n}$

$\frac{3}{2} < \log_2 n$ because $3/2$ is a constant term.

$n^{3/2} < n^{\log_2 n}$ if we raise of to the power on n.

$\therefore$ Now we have $f_3 < f_2 < f_4$

Now comparing with $f_1$ and $f_4$

$f_1 = 2^n$ | $n^{\log_2 n} = f_4$

taking log on both sides.

$n \log_2 2$

$= n$

$\log_2 (n^{\log_2 n})$

$\log_2 n \times \log_2 n$

$= (\log_2 n)^2$

From order of functions we know n grows faster than $(\log_2 n)^2$

$f_1 > f_4$

$\therefore$ The order is $f_3 < f_2 < f_4 < f_1$

---

## 6.12 Solved Problems: Gate 2003. Inversions

In a permutation $a_1, a_2, \ldots a_n$, of n distinct integers, an inversion is a pair $(a_i, a_j)$ such that $i<j$ and $a_i > a_j$. What would be the worst case time complexity of the Insertion Sort algorithm, if the inputs are restricted to permutations of $1 \ldots n$ with at most n inversions?

Ph: 844-844-0102

A. $\theta(n^2)$

B. $\theta(n \log n)$

C. $\theta(n^{1.5})$

D. $\theta(n)$

Ans

Given array of size n.

let us consider the following example of integers {1,2,3,4,5}.

2,1,3,4,5 → (2,1) - 1 inversion

2,3,1,4,5 → (2,1) (3,1) 2 inversions.

2,3,4,5,1 → (2,1) (3,1) (4,1) (5,1) 4 inversions.

The no of swaps = no of inversions in insertion sort

Each of the inversion will make a swap as we move from right to left.

And for each inversion a constant time is required.

∴ We know as it is given the no of inversions are n in the worst case, we have max no of swaps also n, so the worst case time complexity $O(n)$.



Mail: Gatecse@appliedcourse.com

# 6.13 Solved Problem GATE 2003



1. $(n+k)^m = O(n^m)$ k and m are constants
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$

Which of the above statements are true?

Ans.

(1) $(n+k)^m = {}^mC_0 n^m + {}^mC_1 n^{m-1} k^1 + {}^mC_2 n^{m-2} k^2 + \dots$

$\dots {}^mC_m k^m$

$= n^m + C_1 n^{m-1} + C_2 n^{m-2} + \dots C_m n^0$

$= O(n^m)$ 1 is true.

(2) $2^{n+1} = 2 \cdot 2^n$ $\quad 2 \cdot 2^n \leq c \cdot 2^n \quad c = 3 \quad \forall n \geq n_0.$

$n_0 = 1.$

$= O(2^n)$ 2 is also true.

(3) $2^{2n+1} \leq c \cdot 2^n \cdot \exists c, n_0$.
$\forall n \geq n_0$.

$2 \cdot 2^{2n} \leq c \cdot 2^n$.

If we take $c=2$.

$2^{2n} \leq 2^n$

taking log on both sides.

$2n \leq n$ × it does not hold for any $c, n_0$.

let us take $c = 3$ . . .

$2^{2n+1} \leq 3 \cdot 2^n$

$2 \cdot 2^{2n} \leq 3 \cdot 2^n$

$2^{2n} \leq \frac{3}{2} 2^n$

taking log on b.s.

$2n \leq \log_2(3/2) + n$.

It does not hold for large values of $n$ if we take $n > \log_2 3/2$

$\therefore$ statement 3 is false.

# 6.14 Solved Problem : GATE 2008



Q) Consider the following functions

$f(n) = 2^n$

$g(n) = n!$

$h(n) = n^{\log n}$

Which of the following statements about the asymptotic behaviour of $f(n)$, $g(n)$, $h(n)$ is true?

A. $f(n) = O(g(n))$; $g(n) = O(h(n))$

B. $f(n) = \Omega(g(n))$; $g(n) = O(h(n))$

C. $g(n) = O(f(n))$; $h(n) = O(f(n))$

D. $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$

Ans

We know that $g(n) = n! \geq f(n) = 2^n$. from order of functions.

comparing $n^{\log n}$ with $f(n)$

| $n^{\log n}$ | $2^n$ |
|---|---|
| log on both sides. | |
| $\log n \times \log n$ | $n$. |
| $(\log n)^2$ | $n$ |

We know that $(\log n)^2 < n$.

$\therefore h(n) < f(n) < g(n)$.

From this only D is correct

# 6.15 GATE 2013



8)
```
int unknown (int n)
{
    int i, j, k = 0;
    for (i = n/2; i <= n; i++)
    {
        for (j = 2; j <= n; j++)
        {
            k = k + n/2
        }
    }
    return (k);
}
```

What is the return value of this function?

(a) $\Theta(n^2)$

(b) $\Theta(n^2 \log n)$

(c) $\Theta(n^3)$

(d) $\Theta(n^3 \log n)$

Ph: 844-844-0102

Ans.

- Outer loop runs = $n/2$ times ($i = n/2$ to $n$, $i$ incremented every time).

- ~~times~~ Inner for loop runs = $\lfloor \log_2 n \rfloor$ times because $j$ increases from $j=2$ upto $n$ every time it is multiplied by 2.

$2, 4, 8, 16 \ldots\ldots 2^k \ldots\ldots n.$

$n = 2^m$ (lets say)

$m = \log_2 n$. it executes $\log_2 n$ times.

- The value of $k$ is incremented by $n/2$ each time the inner loop is executed.

$\therefore$ Inner loop is executed $\underbrace{n/2}_{\text{\# of times outer loop is executed}} \times \lfloor \log_2 n \rfloor$ times

$\therefore$ Value of $k = n/2 \times \lfloor \log_2 n \rfloor \times \frac{n}{2}$

$= \frac{n^2}{4} \times \lfloor \log_2 n \rfloor$

$= O(n^2 \log_2 n)$



Mail: Gatecse@appliedcourse.com

6.16. GATE 2006



Q) Consider the following C-program fragment in which i, j and n are integer variables.

for (i=n, j=0; i>0; i/=2, j+=i);

Let val(j) denote the value stored in the variable j after termination of the for loop. Which of the following is true?

A. $val(j) = \theta(\log n)$

B. $val(j) = \theta(\sqrt{n})$

C. $val(j) = \theta(n)$

D. $val(j) = \theta(n \log n)$

Ans.

- i takes values $n, n/2, n/4, \ldots 1$

It executes $\log_2^n$ times

The value stored in j is the sum of all values of i.

$$j = ~~1+2+4~~ \; n + \frac{n}{2} + \frac{n}{4} + \cdots \frac{n}{2^i} + \cdots + 1$$

$$= n\left[1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \frac{1}{2^{\log_2 n}}\right] \quad (1)$$

↳ This is a G.P. (Geometric Progression).

a) The inner bracket $< \left(1 + \frac{1}{2} + \frac{1}{4} + \ldots \infty\right)$ if we consider infinite series

our bracket is a finite series is allways < infinite series of same kind

For $1 + \frac{1}{2} + \frac{1}{4} \ldots \infty$

$= \frac{a}{1-r} = \frac{1}{1-\frac{1}{2}} = \underline{2}$ if we remove 1 on L.H.S

$\Rightarrow$ sum is 1

$\therefore$ our series is < the infinite series.

$\therefore$ Eq (1) can be re written as

$< n(1+1)$

$\Rightarrow < n \times 2$

$\underline{O(n)}$ option c is correct.

# Topic 7 Merge Sort

## Video 7.1. Why Lern Another Sorting Algorithm

- Next sorting algorithm is merge sort; but why should we learn another sorting algo when we know one?

Insertion Sort — Worst Case $O(n^2)$
Best Case $O(n)$

- Merge Sort has a worst case time complexity of $O(n \log n)$ which is faster than $O(n^2)$.

- Merge Sort uses divide & conquer algorithm design strategy.

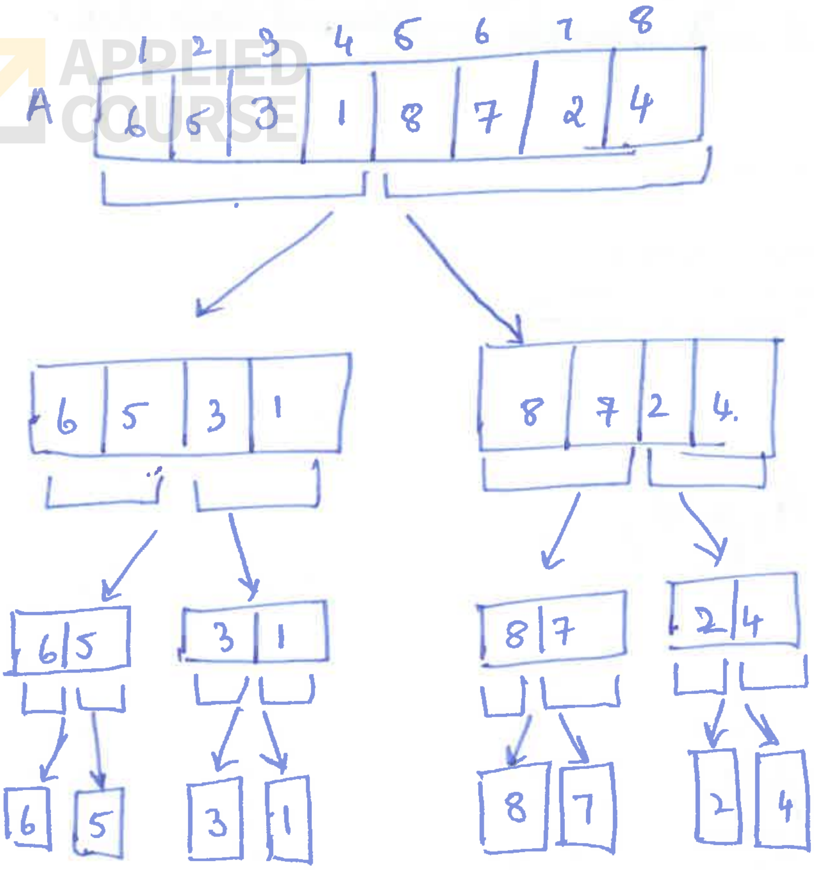
- Merge Sort can be used to sort v.v.v. large amount of data even if the complete data cannot fit into our RAM.

## 7.2. How It Works: Intuition

A

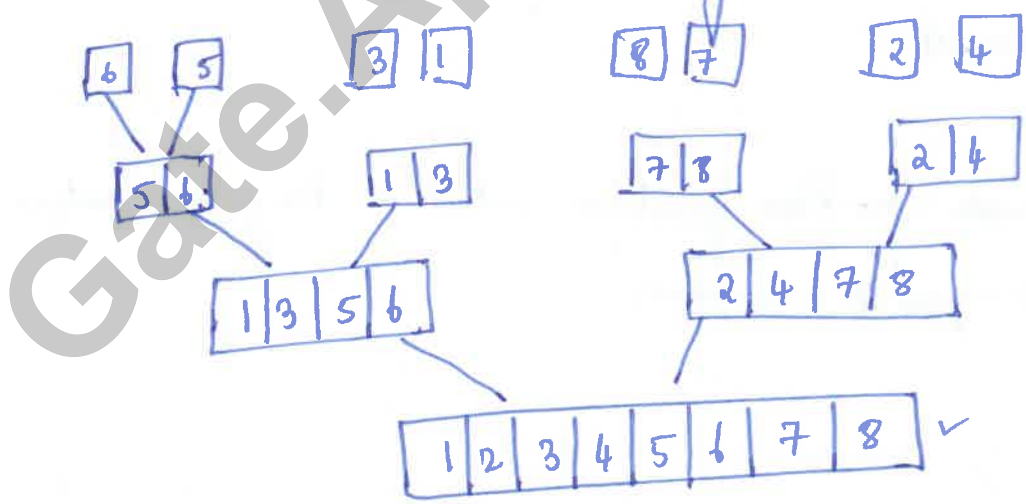
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 5 | 3 | 1 | 8 | 7 | 2 | 4 |

— divide into 2 parts




→ This is known as DIVIDE stage the array is divided into two parts at every stage until a single element is left

→ Now we have the MERGE Or conquer stage where 2 elements or array are MERGED / into one sorted array



→ A big problem is divided into smaller problems at each stage.




## 7.3 Pseudo Code

MERGE-SORT(A, p, r)

1. if $p < r$
2. $q = \lfloor (p+r)/2 \rfloor$
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, q+1, r)
5. MERGE(A, p, q, r)

→ $p < r$ is true if the size of the array > 1. if $p = r$ then there is only one element.

→ $\lfloor \; \rfloor$ stands for floor function which is the greatest integer less than or equal to a number.

# MERGE (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. create arrays $L[1 \ldots n_1+1]$ and $R[1 \ldots n_2+1]$
4. for $i \leftarrow 1$ to $n_1$
5. do $L[i] \leftarrow A[p+i-1]$
6. for $j \leftarrow 1$ to $n_2$
7. do $R[j] \leftarrow A[q+$
8. $L[n_1+1] \leftarrow \infty$
9. $R[n_2+1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. for $k \leftarrow p$ to $r$
13. do if $L[i] \leq R[j]$
14. then $A[k] \leftarrow L[i]$
15. 16. $i \leftarrow i+1$
16. else $A[k] \leftarrow R[j]$
17. $j \leftarrow j+1$

→ The MERGE-SORT function divides the array into two subarrays and recursively calls itself. this happens until $n > 1$.

→ The MERGE is called once MERGE-SORT is completed.

→ The Merge is meant for merging two sorted subarrays into one sorted subarray.

→ lines 1 and 2 calculate the size of the 2 subarrays.

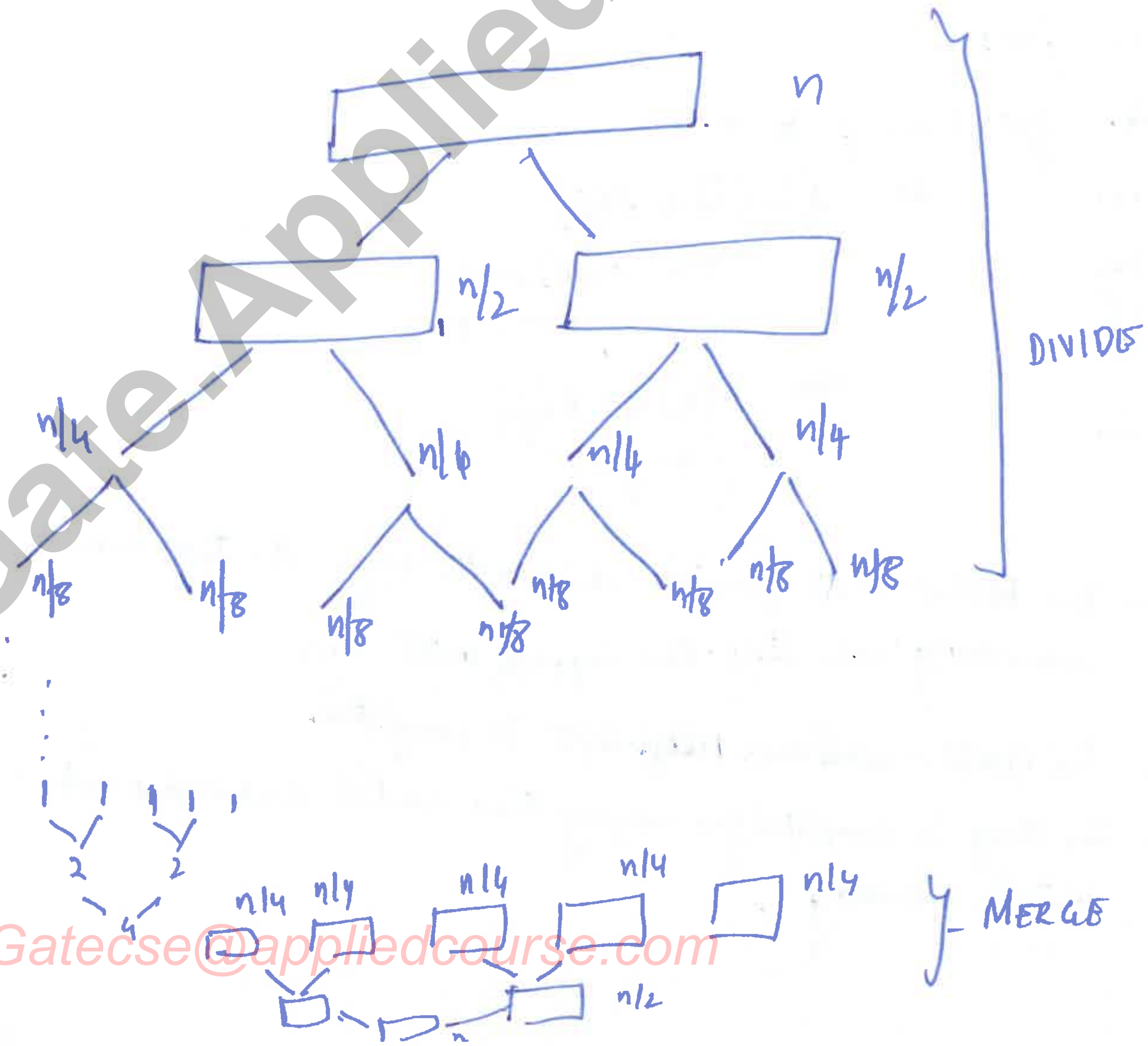
→ loop in lines 4 to 7 copy the elements to new arrays.

→ In loop from lines 12 to 17 :-

- Elements from both sub arrays are compared the one which is small is first copied to the original main array A, then depending on the one smaller element either from the first or second sub array, finally the resulting array is sorted.

## 7.4 Analyzing Time And Space Complexity

→ If T(n) is the time required to sort n elements



The main array of size n is broken down into 2 arrays of size n/2

- n/2 – broken down to 2 × n/4

n/4 – " " 2 × n/8.

This is repeated until we have only one element arrays.

– Then we perform the MERGE operation after the MERGE SORT is completed.

$$T(n) = 2 \times T(n/2) + T(\text{Merge 2 arrays of size } n/2)$$

At a lower level.

$$T(n/2) = 2T(n/4) + T(\text{Merge 2 arrays of size } n/4).$$

Time complexity for size n array for Merge function:

- Lines 1, 2, 3 – are constant time
- loop of line 4–5, 6–7 the sum of the complete no of times the loop will be equal to the total no of elements in the array one level above it.
- At the first level. the time will be $n/2 + n/2 = O(n)$
- At the second level the time will be $n/4 + n/4 + n/4 + n/4 = O(n)$
- At the third level the time would be $n/8 \times 8 = O(n)$.

∴ At each level the time is $O(n)$.

The recurrence relation of Merge Sort can be written as



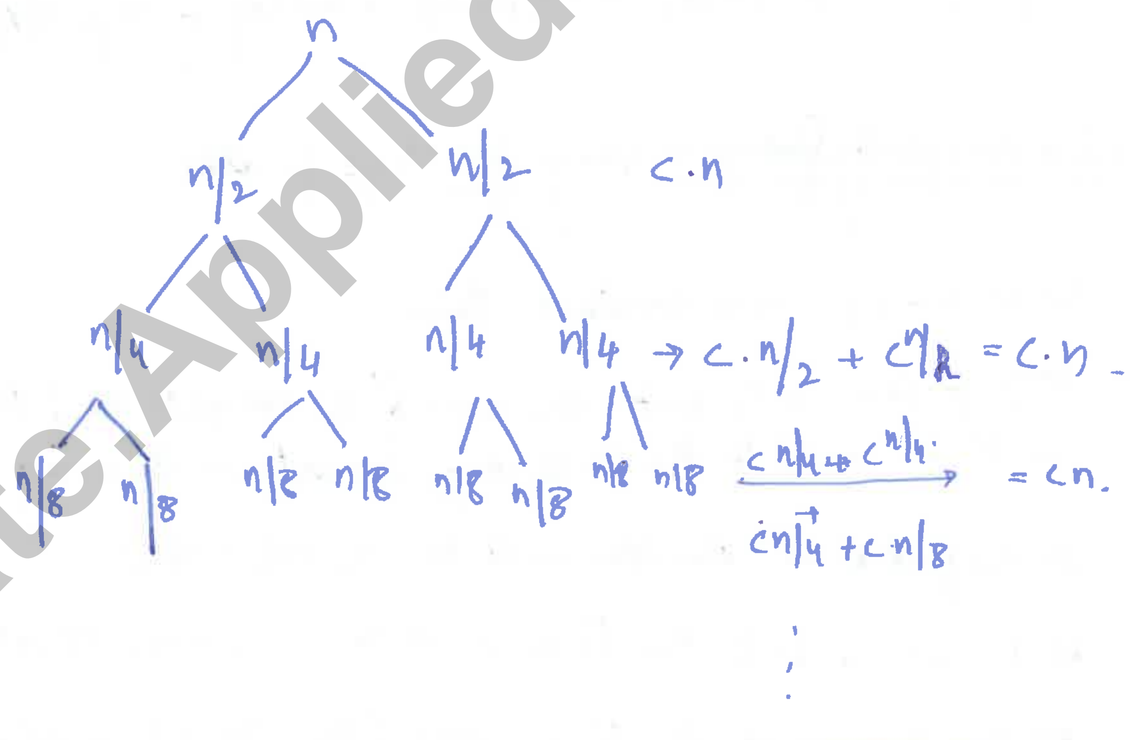
$$T(n) = T(n/2) + T(n/2) + O(n)$$

$$T(n) = 2 \times T(n/2) + O(n)$$

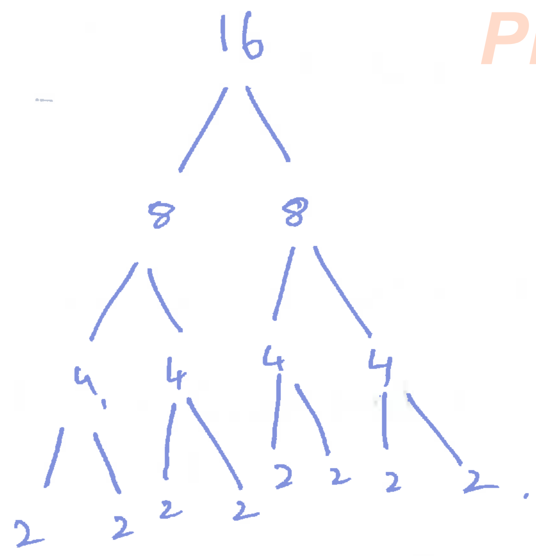
The space complexity of an array of size n $S(n) = O(n) + \text{constant}$
As additional n elements space is used.

# 7.5 Recursion Tree Method.

## Intuition




What is the total time complexity ?



$n = 8$ — 3 levels. $\rightarrow$ $3 \cdot c \cdot n$

$n = 16$ — 4 levels. $\rightarrow$ $4 \cdot c \cdot n$

$n = 32$ — 5 levels $\rightarrow$ $5 \cdot c \cdot n$.

$\log_2 n$ levels. $\log_2 16 = 4$

$\log_2 8 = 3$.

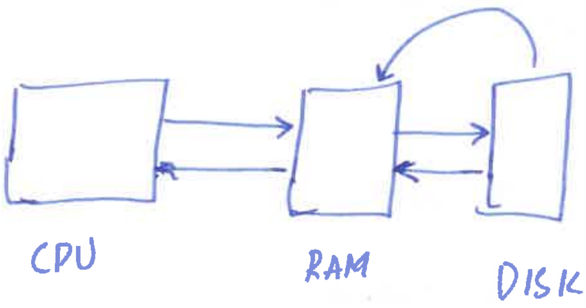
$\therefore$ At Each level we have $O(n)$ work being done and there are $\log_2 n$ levels.

$\therefore$ Time complexity $O(n \log n)$

Space complexity $O(n)$

# 7.6. External Merge Sort

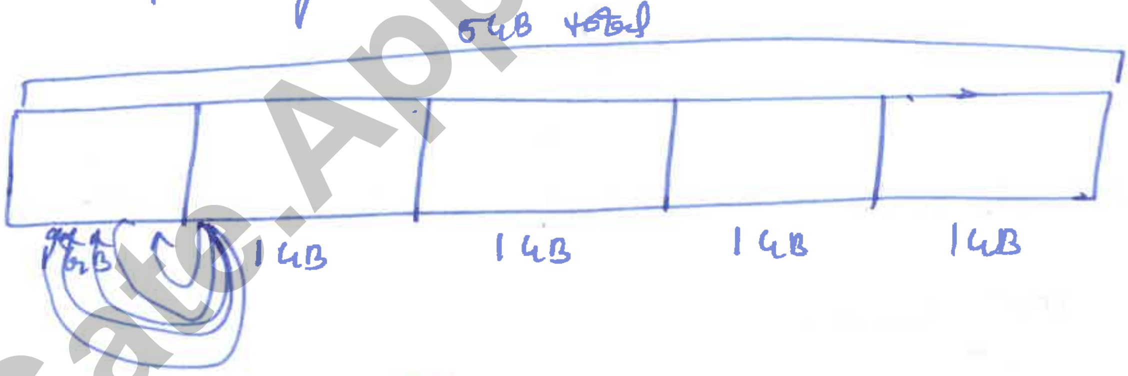
- If we have 5 GB of data on the disk in the form of files which needs to be sorted and our RAM capacity is 1 GB capacity




(Data has to be loaded into the RAM from the DISK)

- Insertion sort cannot be used in such a case because it requires the complete array to be sorted to be present in the memory at once.

→ Let us picturize the 5GB data as below.




1st 1GB is loaded initially and sorted

Next 1GB 1st element has to be compared with the 1st 1GB RAM elements

that is why insertion sort does not work.

In case of external Merge Sort.

| | | |
|---|---|---|
| 5 GB | 1GB ($F_1$) It is loaded into the RAM and sorted | $F_1$-S |
| | 1GB ($F_2$) | $F_2$-S |
| | 1GB ($F_3$) | $F_3$-S |
| | 1GB ($F_4$) | $F_4$-S |
| | 1GB ($F_5$) | $F_5$-S |

similarly remaining 4 files are loaded into the RAM sorted and then put back on the drive.

- Now we can load 150 MB of data from each sorted file $F_i$-S into the RAM

| | | | | | |
|---|---|---|---|---|---|
| $F_1$-S 150MB | $F_2$-S 150MB | $F_3$-S 150MB | $F_4$-S 150MB | $F_5$-S 150MB | |

Remaining 250 MB.

Total 1GB

→ Now this 250 MB is utilized to store the merged into one array. once 250 MB is full it can be moved onto the disk, then this process can be repeated until any of the I/p files is completely processed then the next 150 MB chunk can be used from that file.

- In this way finally we will have multiple files which are O/p files having elements which are sorted.

- This is why/how Merge sort helps in sorting arrays which are larger than the RAM size.

- It was designed in 1940's but even today its used, we have large RAM but the size of the data has also increased enormously.

---

## 7.7 Solved Problem GATE 2007

Q)

```
int j, n;
j=1;
while (j<=n)
    j=j*2;
```

(a) $\lceil \log n \rceil + 2$.

(b) $n$

(c) $\lceil \log n \rceil$

(d) $\lfloor \log n \rfloor + 2$.

$n>0$ How many comparisons are made in the while loop?

<u>Ans.</u> The comparisons are only done in the condition of the while loop.

lets say $n=5$

$j=1$

$j=2$ 4 comparisons.

$j=4$

$j=8$ ← condition fails

$n = 8$

$j = 1$

$j = 2$

$j = 4$

$j = 8$

$j = 16$ — condition fails.

5 comparisons

Checking options (a) $\lceil \log_2 5 \rceil + 2$

$\lceil 2.- \rceil + 2$

$3 + 2 = 5$ ✗

(b) $n = 5$ not correct ✗

(c) $\lceil \log_2 n \rceil = \lceil \log_2 8 \rceil = 3$ not correct ✗

(d) $\lfloor \log_2 n \rfloor + 2 = 2 + 2 = 4$ comparisons ✓

d is the correct options

## 7.8 Solved Problem

Q) sum = 0;

```
for (i=1; i<=n; i=i+1)
{
    for (j=1; j<=n; j=j*2)
    {
        sum = sum + j;    # of times this line is executed?
    }
}
```

Ph: 844-844-0102

Ans

$i \rightarrow 1, 2, 3, 4 \ldots\ldots n.$

for each value of $i$

$j = 1, 2, 4, 8 \ldots\ldots n.$

$2^1 \quad 2^2 \quad 2^3$

It will run $O(\log_2 n)$ no of times.

$\therefore$ The outer loop is executed $n$ times it will execute $O(n \log n)$ times.

---

## 7.9 Solved Problem GATE 2016

Q) Assume Merge sort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of the problem that can be solved in 6 minutes.

A. 256
B. 512
C. 1024
D. 2018.


Mail: Gatecse@appliedcourse.com

Ans

We know worst case. Merge sort $\Theta(n \log n)$.

$n = 64 \longrightarrow 30$ sec

$c \cdot n \log_2 n = 30$

$c \cdot 64 \log_2 64 = 30$

$$c = \frac{30}{64 \times 6}$$

Now 6 mins $= 6 \times 60 = 360$ sec

$c \times m \log m = 360$

$\frac{30}{64 \times 6} \times m \log m = 360$

$m \log m = 4608$

on checking option A. $256 \times \log 256$

$= 256 \times 8$

$= 2048$

option B $= 512 \log_2 512$

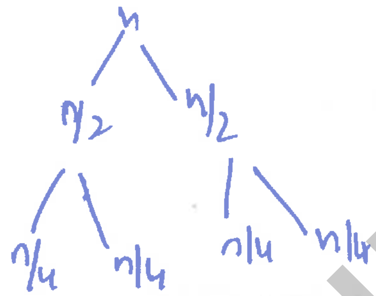
$= 512 \times 9$

$= 4608$ option B is correct.

# Topic : Solving Recurrences

## 9.1 Recursion Tree Method

Merge-Sort :- $T(n) = 2T(n/2) + cn$ ← recurrence relation

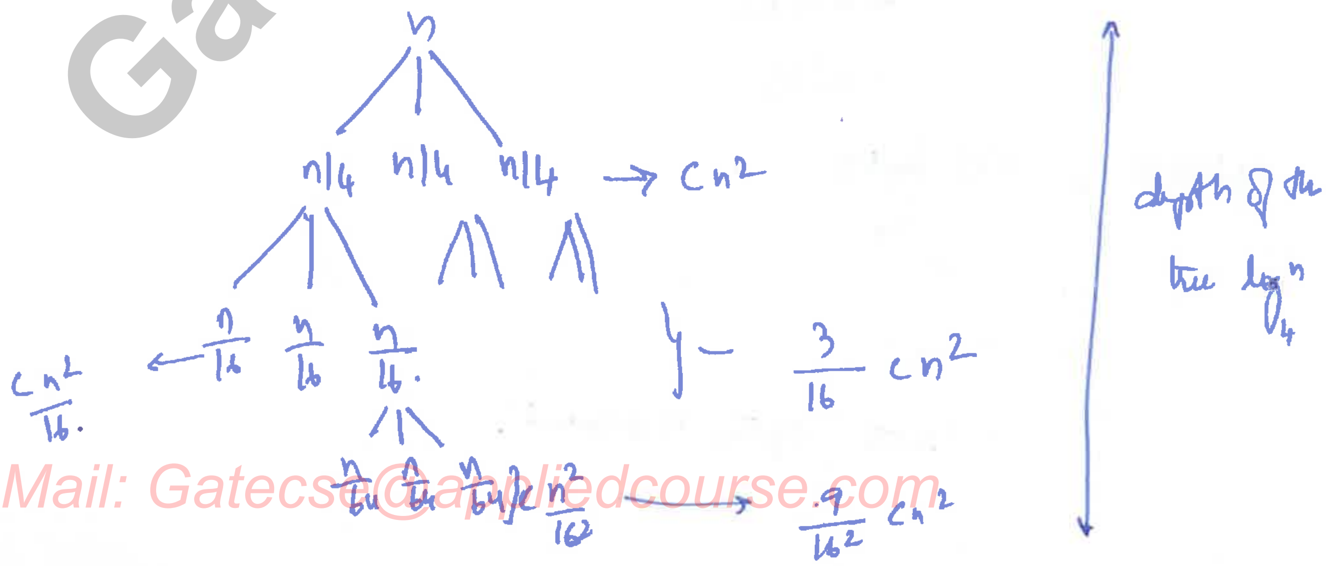



Why should we care about recurrence relations?

⇒ Because they occur many times, whenever we have a recursive algorithm we will have time complexity in the form of a recurrence relation only.

– Also studied in discrete mathematics

$T(n) = 3T(n/4) + cn^2$



1st level $cn^2$

2nd level $\frac{3}{16}cn^2$

3 rd level $\left(\frac{3}{16}\right)^2 cn^2$

$\left(\frac{3}{16}\right)^3 cn^3$

$\vdots$

Total Sum $= cn^2\left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \dots\right]$.

Geometric Progression

$r < 1 \quad r = \frac{3}{16}$

$$T(n) = cn^2\left[\frac{1}{1-\frac{3}{16}}\right]$$

(considering infinite series it will be less than the infinite series).

$$= cn^2\left[\frac{16}{13}\right]$$

$$= \underline{\underline{O(n^2)}}$$

# 9.2. MASTER THEOREM

If we have a recurrence relation of the form.

$$T(n) = aT(n/b) + f(n) \quad a \geq 1, b > 1$$

Case 1: if $f(n) = O\left(n^{\log_b^a - \epsilon}\right)$ for some $\epsilon > 0$.

then $T(n) = \theta\left(n^{\log_b^a}\right)$

example. $T(n) = 9T\left(\frac{n}{3}\right) + n$.

$a = 9 \quad b = 3 \quad f(n) = n$.

$\log_b^a = \log_3^9 = 2$.

$$O\left(n^{\log_3^9 - \epsilon}\right) = O\left(n^{2-\epsilon}\right)$$

$\epsilon = 1$

$$O(n^1)$$

soln = $T(n) = \theta\left(n^{\log_b^a}\right) = \theta\left(n^2\right)$

Case 2: If $f(n) = \Theta\left(n^{\log_b^a}\right)$

then $T(n) = \Theta\left(n^{\log_b^a} \cdot \log n\right)$.

Example

$$T(n) = 1 \cdot T\left(\frac{2n}{3}\right) + 1 \qquad a = 1, b = \frac{3}{2} \quad f(n) = 1$$

$$n^{\log_b^a} = n^{\log_{3/2}^1} = n^0 = 1 \quad (\text{it satisfies case 2})$$

$$\therefore T(n) = \Theta\left(n^{\log_b^a} \log n\right) = \Theta\left(n^{\log_{3/2}^1} \cdot \log n\right)$$

$$= \Theta(\log n)$$

Case 3

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad a \geq 1, b > 1$$

If $f(n) = \Omega\left(n^{\log_b^a + \epsilon}\right) \quad \epsilon > 0$.

AND $a f\left(\frac{n}{b}\right) \leq c f(n)$ ; $c < 1$ $\forall n$.

Example:

$T(n) = 3T\left(\frac{n}{4}\right) + n\log n.$

$a=3 \quad b=4 \quad f(n) = n\log n \qquad \log_4 3 = 0.793$

If $f(n) = n\log n = \Omega\left(n^{0.793+\epsilon}\right)$ $\quad \epsilon \simeq 0.2.$

$0.793+.2$

$\simeq 1$

$n\log n = \Omega(n).$

If $3 \cdot \frac{n}{4} \log\frac{n}{4} \leq c\, f(n)\ \forall n \ ;\ c<1.$

$\frac{3}{4} n \log\frac{n}{4} \leq \frac{3}{4} n\log n.$

$T(n) = \Theta(f(n))$

$= \underline{\underline{\Theta(n\log n)}}.$

## 9.3 Extended Master Theorem

$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n) \qquad a \geq 1 \quad b>1, k>0$ p is a real no.

① If $a > b^k$. then $T(n) = \Theta\left(n^{\log_b a}\right).$

2) if $a = b^k$ then.

a) $p > -1 \rightarrow T(n) = \theta\left(n^{\log_b^a} \cdot \log^{p+1} n\right)$

b) $p = -1 \rightarrow T(n) = \theta\left(n^{\log_b^a} \log \log N\right)$

c) $p < -1 \rightarrow T(n) = \theta\left(n^{\log_b^a}\right)$

3) if $a < b^k$

a) $p \geq 0 \rightarrow T(n) = \theta(n^k \log^p n)$

b) $p < 0 \rightarrow T(n) = \theta(n^k)$

eg

$T(n) = 2T\left(\frac{n}{2}\right) + n\log^2 n.$

$a = 2 \quad k = 1$

$b = 2 \quad p = 2.$

$T \quad a = b^k$ case 2.

$2 = 2$

$T(n) = \theta\left(n^{\log_b^a} \log^{p+1} n\right) = \theta(n^1 \log^3 n)$

# 9.4. Inadmissible Cases & Shortcuts





Reference link :- https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)#Inadmissible_equations.

- There are failure cases of the master theorem.

1. $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$.

   a is not a constant here.

2. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$ master theorem does not work.

   but Extended master theorem does not work.

   $p = -1$.

3. $T(n) = 0.5\,T\left(\frac{n}{2}\right) + n$. a < 1 cannot have less than 1 for a sub problem.

4. $T(n) = 64\,T\left(\frac{n}{8}\right) - n^2 \log n$

   f(n) which is the combination time is not positive.

5. $T(n) = T\left(\frac{n}{2}\right) + \underbrace{n(2 - \cos n)}_{f(n)}$

If we have functions like sin n and cos n we cannot apply master theorem as their value oscillate and they are not simple monotonic functions.

— Some examples where we can easily solve the recurrence relation

① $T(n) = T(n/2) + 2^n$

$= O(2^n)$

② $T(n) = 2T(n/2) + n!$

$= O(n!)$

— Only for exponential and factorial functions we can directly write this.

---

## 9.5 Substitution Method

Recursion tree, Master theorem, Substitution method is another way.

by way of mathematical induction we can solve it. (MI).

Let us assume $T(n) = 2T(n/2) + n$ — Merge Sort.

Guess :- $T(n) = O(n \log n)$.

Assume that it is true for all $m < n$.

M.I. To prove $T(n) \leq cn \log n$ assuming $T(m) \leq cm \log m$

$\forall m < n$.

if $m = \frac{n}{2} < n$.

assuming $T\left(\frac{n}{2}\right) \leq c\,\frac{n}{2} \log \frac{n}{2}$

$$T(n) = 2T(n/2) + n.$$

$$\leq 2c \cdot \frac{n}{2} \log \frac{n}{2} + n.$$

$$\leq c \cdot n \log \frac{n}{2} + n.$$

$$\leq c \cdot n \left[\log_2 n - \log_2 2\right] + n$$

$$\leq cn\left[\log_2 n - 1\right] + n.$$

$$\leq cn\log n - cn + n.$$

$$\leq \underline{cn\log n}.$$

lets now guess incorrectly

$$T(n) = 2T(n/2) + n.$$

guess $T(n) = O(n) \Rightarrow T(n) \leq cn.$

assume $T(m) = O(m) \leq cm \;\forall\, m < n.$

To prove: $T(n) \leq cn$ assuming $T(m) \leq cm \;\forall\, m < n.$

$$T(n) = 2T(\tfrac{n}{2} + n)$$

$$\leq 2c \cdot \frac{n}{2} + n$$

$$= cn + n$$

$T(n) \leq cn+n$ — we did not get $\boxed{T(n) \leq cn}$

# Topic: Recursion In Programming

## 8.1. Factorial : Time & Space Complexity

→ Recursion : A program / functions call itself.

→ Many Algorithms are recursive and is widely used and is a great idea.

→ Lets consider an example of factorial

$$f(n) = \begin{cases} n \times f(n-1) & \text{otherwise } (n \geq 1) \\ 1 & n = 0 \text{ or } 1 \end{cases}$$

```
f(n)
{
    if (n=0 OR n=1)
        return 1
    else
    {
        return n * f(n-1)
    }
}
```

How is this program executed.

$f(5)$ — 120
↳ $5 \times f(4)$ — 24
↳ $4 \times f(3)$ — 6
↳ $3 \times f(2)$ — 2
↳ $2 \times f(1)$
↳ 1

| | |
|---|---|
| $2 \times \cancel{(.)}$ | $f(1)$ |
| $3 \times \cancel{(.)}$ | $f(2)$ |
| $4 \times \cancel{(.)}$ | $f(3)$. |
| $5 * \cancel{(.)}$ | $f(4)$ |

Call Stack: Keeps track of all the calling functions & variables.

Time Complexity.

$T(n) = T(n-1) + c$.

$n$ — $c$
$n-1$ — $c$
$n-2$ — $c$
$n-3$ — $c$
⋮ ... ...
$1$ — $c$

$n(c) \rightarrow O(n)$

Space Complexity :- The size of the call stack at most is $(n-1) \times c$

$c$ is for each function call. $= \underline{O(n)}$

---

## 9.2. Recursion Vs Iteration



| | |
|---|---|
| f (n)<br>  if n = 0 OR n = 1<br>    return 1<br>  else<br>    return n × f(n-1) | f(n)<br>  p = 1<br>  for i = 1 to n.<br>    p = p * i<br>  return p. |
| $T(n) = c + T(n-1) = O(n)$<br>$S(n) = O(n)$ call stack | $T(n) = O(n)$<br>$S(n) = O(1)$<br>└ only one variable is used (p). |

→ Most of the times if there is an iterative version possible then we should prefer the iterative version so that we do not have the space required for the call stack.

→ Some times we can save space when we convert a recursive algorithm to an iterative algorithm.

→ Tail recursion (call Tail) Helps in such cases.

## 9.3 Tail Recursion / Tail - Call Optimization

f(n)
  if (n=0 OR n>1)
    return 1
  else .
    return n×f(n-1) .

→ Head Recursion

→ After the recursion returns there are still some operations to perform

fTR(n, a) → accumulator
  if (n=0 OR n=1)
    return a .
  else
    return fTR(n-1, n*a)

fTR(5,1)
↓
fTR(4,5)
↓
fTR(3,20)
↓
fTR(2,60)
↓
fTR(1,120).

→ Tail recursion

→ Idea to reduce the space complexity in case of a recursive algorithm.

→ After the recursive function calls to the original calling function no other operations is performed except the return .

→ Most modern compilers (C, C++, java, python) as soon as they detect tail recursion, they convert the recursive code to iterative code so that no space is wasted in call stack:

f TR → (modern compiler) → iterative   space complexity is reduced. O(1)

Recursion

## Topic: Solved Problems Of Solving Recurrences & Recursion In Programming

### 11.1 GATE 2011

Q) Consider the following C recursive function that takes two arguments

```c
unsigned int foo (unsigned int n, unsigned int r) {
    if (n>0) return ((n%r) + foo(n/r, r));
    else return 0;
}
```

What is the return value of the function foo when it is called as foo(345, 10)?

A. 345

B. 12

C. 5

D. 3

Ans. foo(345, 10) 12

↳ 345 % 10 + foo(34, 10)

↳ 34 % 10 + foo(3, 10)

↳ 3 % 10 + foo(0, 10)

Option B 12 is correct.

## 11.2 Fibonacci: Time Complexity

(Q) fib(n)

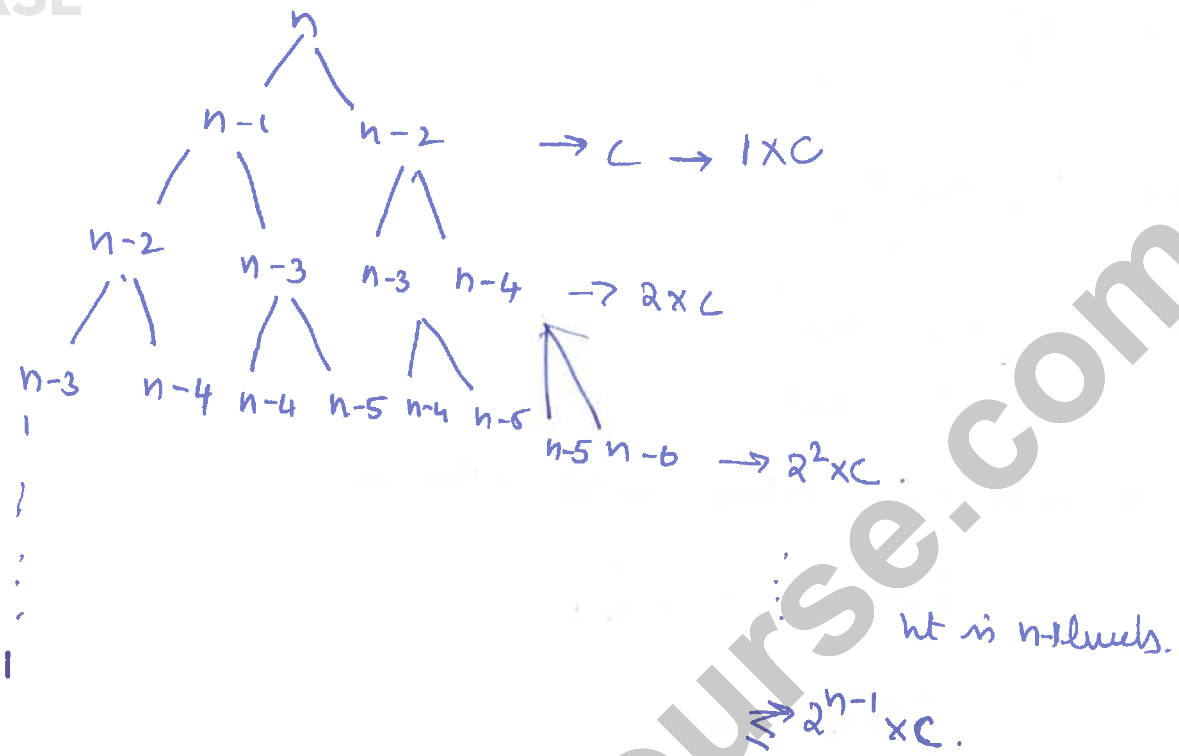
if n=0
    return 0;
if n=1
    return 1;
else
    return fib(n-1) + fib(n-2);

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-1) + fib(n-2) & \text{otherwise.} \end{cases}$$

Above code is not Tail recursive.

$T(n) = T(n-1) + T(n-2) + C$

Recursion tree method




Total time required $\leq c + 2^1 c + 2^2 c + \dots 2^{n-1} \times c$.

$$T(n) \leq c\{1 + 2 + 2^2 + \dots 2^{n-1}\}$$

$$= c\{2^n - 1\}$$

$$= c\{2^n\}$$

$$T(n) \leq 2^n \Rightarrow \underline{\underline{T(n) = O(2^n)}}$$

---



## 11.3 GATE 2007

Q) What is the time complexity of the following recursive function?

```
int DoSomething (int n) {
    if (n<=2)
        return 1;
    else
        return (DoSomething (floor(sqrt(n)) + n);
}
```

A. $\Theta(n^2)$

B. $\Theta(n \log_2 n)$

C. $\Theta(\log_2 n)$

D. $\Theta(\log_2 \log_2 n)$.

Ans the recurrence relation can be written as

$$T(n) = T(\sqrt{n}) + c$$

$$T(\leq 2) = c.$$

$$n$$

$$\downarrow$$

$$\sqrt{n} = n^{1/2} \rightarrow c$$

$$\downarrow$$

$$n^{1/4} \rightarrow c$$

$$\downarrow$$

$$n^{1/8} \rightarrow c$$

$$\vdots$$

$$k.\quad 2 \rightarrow c$$

Total time = k.c.

$$n^{\frac{1}{2^k}} = 2. \qquad \frac{1}{2^k} \log n =$$

$$\log\left[\frac{1}{2^k}\log n\right] = 0.$$

$$-k + \log\log n = 0.$$

$$k = \log\log n \quad \text{option D.}$$

## 11.4 GATE 2006

Q) Consider the recurrence relation

$$T(n) = 2T(\sqrt{n}) + 1 \qquad T(1) = 1$$

Which of the following is true.

A. $T(n) = \Theta(\log\log n)$

B. $T(n) = \Theta(\log n)$

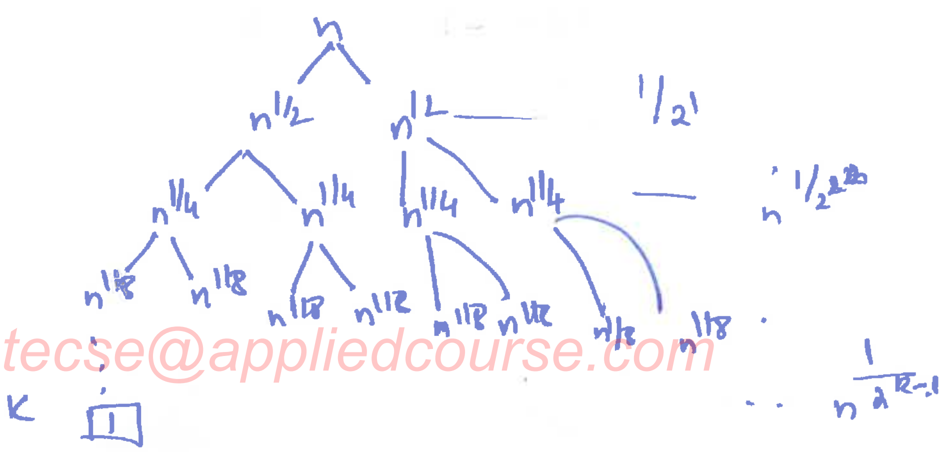
C. $T(n) = \Theta(\sqrt{n})$.

D $T(n) = \Theta(n)$.

Ans

The recurrence relation can be written as

$$T(n) = 2T(\sqrt{n}) + 1$$



Ph: 844-844-0102

The depth of the tree as 8. that of the previous problem, we can get.

$$\frac{1}{2^k} = 2$$

$n$

$$\frac{1}{2^k} \log n = 1$$

$$-k + \log\log n = 0.$$

$$\log\log n = k.$$

Total time complexity. = Sum of all time complexities at each level

$$= c + 2c + 4c + \dots kc.$$

$$\Rightarrow 2^k - 1$$

$$\Rightarrow 2^{\log\log n} - 1$$

$$\Rightarrow \log n - 1$$

Mail: Gatecse@appliedcourse.com

# 11.5 Shortcut + GATE 2004

**Shortcut.**

$$T(n) = aT(n-b) + O(n^k)$$

$a > 0$ ; $b \geq 1$ ; $k \geq 0$

Case 1 if $a > 1$ then $T(n) = O(n^k a^{n/b})$

Case 2 if $a = 1$ then $T(n) = O(n^{k+1})$

Case 3 if $a < 1$ then $T(n) = O(n^k)$

Q. The time complexity of the following C function is (assume $n > 0$).

```
int recursive (int n) {
    if (n==1)
        return(1);
    else
        return (recursive (n-1) + recursive (n-1));
}
```

A. $O(n)$

B. $O(n \log n)$

C. $O(n^2)$

D. $O(2^n)$

Ph: 844-844-0102

Ans.

The recurrence relation can be written as.

$$T(n) = 2T(n-1) + c$$

$a = 2 \quad b = 1 \quad K = 0 \quad \cancel{a=2}$

$$T(n) = O(n^0 \, 2^{n/1})$$

$$= \underline{O(2^n)} \text{ option D.}$$

---

## 11.6 GATE 2009

Q) $T(n) = T(n/3) + \theta(n)$

Ans $\quad a = 1 \quad b = 3 \quad k = 1$.

$a < b^k$ and $p = 0$. case 3 a of extended master theorem.

$$T(n) = \theta(n^k \log^p n)$$

$$= \underline{\theta(n)}$$

---

## 11.7 GATE 2008

Q) $T(n) = \sqrt{2}\, T(n/2) + \sqrt{n}$.

$a = \sqrt{2} \quad b = 2 \quad k = 1/2 \quad p = 0$.

$a = b^k \quad p > -1 \quad$ case 2 (a) of extended master theorem.

$$T(n) = \theta(n^{\log_b^a} \log^{p+1} n) = \theta(n^{1/2} \log n) = \underline{\theta(\sqrt{n} \log n)}$$

Mail: Gatecse@appliedcourse.com

11.8 GATE 2005



Ph: 844-844-0102

Q) $T(n) = 2T(n/2) + n$. Which of the following is false?

(a) $\theta(n \log n)$

(b) $O(n \log n)$.

(c) $O(n^2)$

(d) $\Omega(n^2)$.

Ans It is recurrence relation of Merge Sort $T(n) = \theta(n \log n)$

a is True

b. is also true because $\theta(n \log n)$ means it is both the upper and lower bond of ~~be~~ $T(n)$ it can be written as both $O(n \log n)$ and $\Omega(n \log n)$.

c. $O(n^2)$ is also true because $n^2$ grows faster than $n \log n$ and $n^2$ can be considered as an upper bond for $n \log n$

$\therefore O(n^2)$ is correct.

d. $\Omega(n^2)$ is false because $n^2$ grows faster than $(n \log n)$ and $\Omega$ is a lower bond. $\therefore$ D is correct option.



Mail: Gatecse@appliedcourse.com

11.9 GATE 2003



Q)

$$\begin{cases} T(1) = 1 \\ T(n+1) = T(n) + \lfloor \sqrt{n+1} \rfloor \quad \forall\, n \geq 1 \end{cases}$$

$n+1$
$\downarrow$
$n \rightarrow \lfloor \sqrt{n+1} \rfloor$
$\downarrow$
$n-1 \rightarrow \lfloor \sqrt{n} \rfloor$
$\downarrow$
$n-2 \rightarrow \lfloor \sqrt{n-1} \rfloor$
$\downarrow$
$\vdots$
$1 \longrightarrow \lfloor \sqrt{2} \rfloor$

(a) $\frac{m}{6}\left(\frac{2}{m} - 39\right) + 4$

(b) $\frac{m}{6}\left(4m^2 - 3m + 5\right)$

(c) $\frac{m}{2}\left(3m^{2.5} - 11m + 20\right) - 5$

(d) $\frac{m}{6}\left(5m^3 - 34m^2 + 137m - 104\right) + \frac{5}{6}$

If we consider $T(m^2) = \lfloor \sqrt{m^2} \rfloor + \lfloor \sqrt{m^2-1} \rfloor + \lfloor \sqrt{m^2-2} \rfloor + \cdots \lfloor \sqrt{2} \rfloor + 1$

Easiest way is by method of elimination

for $T(1) = 1$

(a) $T(1) = 1$ (b) 1 (c) 5 (d) 8/3

$\times$ $\times$

(2) $m=2$ $m^2=4$.

$$T(m^2) = \lfloor\sqrt{4}\rfloor + \lfloor\sqrt{3}\rfloor + \lfloor\sqrt{2}\rfloor + 1$$

$$= 2+1+1+1$$

So 1 and 2 is same.

(3) $m=3$ also we get the same.

(4) $m=4$ $m^2=16$.

$$T(m^2) = \lfloor\sqrt{16}\rfloor + \lfloor\sqrt{15}\rfloor + \cdots + \lfloor\sqrt{3}\rfloor + \lfloor\sqrt{2}\rfloor + 1$$

$$T(16) = 4 + (3\times6) + 3 + (2\times4) + (1\times2) + 1 = 38.$$

(a) $\frac{4}{6}(84-39)+4 = 34$

(b) $\frac{4}{6}(64-12+5) = 38$. b is the correct Ans.

---

## 11.10 GATE 2016

(Q) $a_n$ = number of n-bit strings that do NOT contain two consecutive ones. Which of the following are correct

(a) $a_n = a_{n-1} + 2a_{n-2}$

(b) $a_n = a_{n-1} + a_{n-2}$

(c) $a_n = 2a_{n-1} + a_{n-2}$

(d) $a_n = 2a_{n-1} + 2a_{n-2}$

$n=1$ $\{0,1\}$ in such case we have such strings $a_n = 2$

$n=2$ $\{00, 01, 10\}$ $a_2 = 3$.

$n=3$ $\{000, 010, 101, 001, 100\}$ $a_n = 5$

$n = n$

Take all strings of length $(n-1)$ and add "0" at the end.

take all strings of length $(n-2)$ add "01" at the end.

Lets check for $n=4$ $a_n = a_{n-1} + a_{n-2}$.

$n=4$

| | |
|---|---|
| 0000 | 1000 |
| 0001 | 1001 |
| 0010 | 1010 |
| 0011 | 1011 |
| 0100 | 1100 |
| 0101 | 1101 |
| 0110 | 1110 |
| 0111 | 1111 |

- All possible strings

$= \{0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010\}$

$a_4 = 8$.

using the recurrence relation

$a_4 = a_3 + a_2$

$= 5 + 3 = \underline{\underline{8}}$.

for $n=3$ $a_3 = a_2 + a_1$

$= 3+2$

$= \underline{5}$

$\therefore$ option b is correct

11.11. GATE 2016



8) $a_n = 6n^2 + 2n + a_{n-1} \quad n > 1$

$a_1 = 8$

$a_{99} = K \times 10^4$ Find K.

Ans

We can re write as

$a_n = 6n^2 + 2n + a_{n-1}$ for $n \geq 1$

$a_0 = 0$

$a_{n-1}$ can be rewritten as $6(n-1)^2 + 2(n-1) + a_{n-2}$

$a_n = 6n^2 + 2n + 6(n-1)^2 + 2(n-1) + a_{n-2}$

$a_{n-2}$ can be rewritten as $6(n-2)^2 + 2(n-2) + a_{n-3}$

By Mathematical Induction we can write $a_n$ as.

$a_n = 6\{n^2 + (n-1)^2 + (n-2)^2 + \cdots 1^2\}$

$+ 2\{n + (n-1) + (n-2) + \cdots 1\}$

$+ 0$

Ph: 844-844-0102

$$\Rightarrow \frac{6\,(n)(n+1)(2n+1)}{6}.$$

$$+ \frac{2\,(n(n+1))}{2}$$

$$+0$$

$\Rightarrow$ For $n = 99$

$\Rightarrow$ $99(100)(199) + 99(100)$

$\Rightarrow$ $2 \times 99 \times 10^4$

$\Rightarrow$ $198 \times 10^4$

$\underline{\underline{K = 198}}$.



Mail: Gatecse@appliedcourse.com

# Additional Problems for time complexity analysis. (Not in Videos).

Problems are based on determining the time complexity of code segment.

- main ABC )

```
{
  ____
  ____
  ____
  ____
}
```

**NOTE**

Some lines of code without any loop.

then always this is constant time complexity $\underline{O(1)}$

Q Fun( )

```
{
    int i;
    for (i=1 to n)
    {
        printf("Hello")
    }
}
```

- The above loop is executed n times within it we have constant time statements

T. Complexity $O(n \times 1) = \underline{O(n)}$

Q.

```
Function(n)
{
    i=1, sum=0
    for(i=1; sum=0; sum<=n; i++)
    {
        i++;
        sum = sum + i;
        printf("Hello Algorithms");
    }
}
```

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of iterations | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | |

It is ~~six~~ adding up the sum of n natural numbers.

If the loop executes p times $p <= \frac{n(n+1)}{2}$.

$\therefore P = O(\sqrt{n})$

It gets executed $\sqrt{n}$ times and time complexity $O(\sqrt{n})$.

→ Fun displayStars(n)

```
{
    for (i=1; i<=n; i++)
    {
        for (j=1; j<=i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
}
```

Predict O/P of above program and time complexity

- for n=3

```
*
**
***
```

It prints triangular shaped star pattern

The outer loop executes n times for a given n the inner loop executes

n = 5, i=1, -1 time $= 1+2+3+4+5 = \sum n = \frac{n(n+1)}{2} = O(n^2)$ times

i=2 = 2 times

i=3 3 time

i 4 - 4

5 - 5

Q→ Function(n)

```
Function(n)
{
    for (i=1; i<=n ; i=i*2)
    {
        for (j=1 ; j<=n; j++)
        {
            for (k=n ; k>=1 ; k=k/2)
            {
                printf("This is the inner most loop");
            }
        }

        for (m=1; m<=100; m++)
        {
            printf("Hello");
        }
    }
}
```

- loop with k counter is executed $O(\log n)$ times as each time we are dividing by 2 $n \to n/2 \to n/4 \to n/8 - - - - 1$

  $\longleftarrow$ $\log n$ steps $\longrightarrow$

- The loop with index j is executed n times.

  Total time complexity of j and k loops together $= O(n \log n)$

- The loop with index m is executed 100 times which is a constant but we also have it within a loop of index i, which executes $\log n$ times

time complexity for this is $= O(100 \log n) = O(\log n)$.

- The loop with index j is in another loop of index i which executes $\log n$ times

$\therefore$ time complexity for this part $= O(\log n \times n \log n) = O(n \log^2 n)$

$\therefore$ Total time complexity $= O(\log n) + O(n \log^2 n) = \underline{\underline{O(n \log^2 n)}}$.

# Topic : Bubble Sort



## 12.1 How It Works : Intuition + Code

| 3 | 38 | 5 | 44 | 15 | 36 | 47 | 26 | 27 | 2 | 46 | 4 | 19 | 50 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

```
1.    do {
2.        swapped = false;
3.        for i=1 to IndexOfLastUnsortedElement -1
4.
5.            {
6.                if (leftElement > rightElement)
7.                {
8.                    swap(leftElement, rightElement)
9.                    swapped = true
10.               }
              }
11.   } while(swapped);
```



- The loop lines 3-10 every element is compared with its next element if it is not in the correct order then they both are swapped.

- At the end of the 1st iteration of the largest element of the array reaches at its correct location.

- Similarly at the end of the second iteration the 2nd largest element reaches the correct location.

- At the end of the ith iteration the ith element reaches the correct position.

Initially we have

swapped.

3, 38, 5, 44, 15, 36, 47, 26, 27, 2, 46, 4, 19, 50, 48.

i

3, 5, 38, 44, 15, 36, 47, 26, 27, 2, 46, 4, 19, 50, 48.

i i Swapped.

swapped

3, 5, 38, 15, 44, 36, 47, 26, 27, 2, 46, 4, 19, 50, 48.

i

3, 5, 38, 15, 36, 44, 47, 26, 27, 2, 46, 4, 19, 50, 48.

i i

3, 5, 8, 15, 36, 44, 26, 47, 27, 2, 46, 4, 19, 50, 48.

i

3, 5, 8, 15, 36, 44, 26, 27, 47, 2, 46, 4, 19, 50, 48.

i

3, 5, 8, 15, 36, 44, 26, 27, 2, 47, 46, 4, 19, 50, 48.

i

3, 5, 8, 15, 36, 44, 26, 27, 2, 46, 47, 4, 19, 50, 48

i

3, 5, 8, 15, 36, 44, 26, 27, 2, 46, 4, 47, 19, 50, 48.

3, 5, 8, 15, 36, 44, 26, 27, 2, 46, 4, 19, 47, 50, 48.

3, 5, 8, 15, 36, 44, 26, 27, 2, 46, 4, 19, 47, 48, 50.

One Iteration is completed and at end of it we have 50 i.e. the largest element at its correct position.

→ For the second iteration we have

3, 5, 8, 15, 36, ~~44~~ 26, ~~27~~, ~~2~~, ~~46~~, 4, ~~19~~, 47, 48, 50

26, ~~44~~ 27   2   44   4, ~~46~~. 19   46.

→ At the end of the second loop we have the second largest element present at its correct position.

At the end of second iteration we have

3, 5, 8, 15, 36, 26, 27, 2, 44, 4, 19, 46, 47, 48, 50.

*** → The purpose of the swap variable is to keep track of the swaps. If there is no swap in a particular iteration then we can say that it is already sorted and no need to sort further and the loop is omitted.

- At end of 3rd Iteration we have.

3, 5, 15, 36, 26, 27, 2, 38, 4, 19, 44, 46, 47, 48, 50.

→ Similarly at end of it we get the sorted array

2, 3, 4, 5, 15, 19, 26, 27, 36, 38, 44, 46, 47, 48, 50.

→ Reference link for visualisation https://visualgo.net/en/sorting.

## 12.2 Space And Time Complexity

→
```
1.  do
2.    {
3.        swapped = false.
4.        for i = 1 to noOfUnsortedElements - 1
5.        {
6.            if (leftElmt > right Elmt)
7.            {
8.                swap( leftElmt, rightElmt)
9.                swapped = true.
10.           }
11.       }
12.   } while (swapped).
```

→ In the worst case the loop 4–12 will execute n times

i = 1   n comparison   n swaps

= 2   n-1 comparison   n-1 swaps

3   n-2 comparisons.   n-2 swaps.

⋮

(n-1)   1 compari   1 swap

= 1+2+3 ... +n =

⇒ $O(n^2)$ swaps.

$O(n^2)$ comparisons.

In the worst case the time complexity $= O(n^2)$ (when array is in descending order)

- In the best case, it occurs when the array is already sorted.

We have one pass through the complete array and no swaps,

# of comparisons $O(n)$ and no. of swaps $= 0$ $= O(1)$.

$\therefore$ Time complexity : Worst Case $O(n^2)$

Best Case $O(n)$

Avg Case $(n^2)$ (when partially sorted).

<u>Space Complexity</u> :- $O(1)$ because this is an inplace sorting algorithm, only 2 additional variables are used.

---

## <u>12.3 Why should bubble sort be avoided?</u>

- Even though the asymptotic time complexity is same as that of insertion sort, experimental results show the number of comparisons which are required by bubble sort are very large.

→ Bubble sort does not have many applications in real world.

→ Many researchers argue that it should not be part of the academic curriculum.

## 12.4 Solved Problem Gate 2000

Q) An array contains four occurances of 0, five occurences of 1, and three occurances of 2 in any order. The array is to be sorted using swap operations (elements that are swapped need to be adjacent).

1. What are minimum no of swaps needed to sort such an array in the worst case.

2. Given an ording of elements in the above array so the minimum no of swaps needed to sort the array is maximum.

Ans

0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2 in any order.

Adjacent elements are swapped only in bubble sort.

1) Worst Case:

2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0.

↑ ↑ ↑ ⟵⟶ ⟷

.................... - 9 swaps.

(9 + 9 + 9) for 2's.

the array is now

1, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2.

Now while swapping 1's = 4 + 4 + 4 + 4 + 4 swaps

= 4 x 5

∴ Total no of swaps = 9 x 3 + 4 x 5

= 27 + 20

= 47

(Q) The ordering is the array in descending order

2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0.

---

## 12.5 Gate Problem GATE 1995

Q) Consider the following sequence of numbers.

92, 37, 52, 12, 11, 25.

Use bubble sort to arrange the sequence in ascending order. Give the sequence at the end of each of the first five passes.

Ans

1st Iteration

$\cancel{92}, \cancel{37}, \cancel{52}, \cancel{12}, \cancel{11}, \cancel{25}$

37 . 52 12 11 25 92

37, 52, 12, 11, 25, 92

2nd Iteration

37, $\cancel{52}$, $\cancel{12}$, $\cancel{11}$, $\cancel{25}$, 92

12 11 25 52

37, 12, 11, 25, 52, 92

3rd Iteration

$\cancel{37}$, $\cancel{12}$, $\cancel{11}$, $\cancel{25}$, 52, 92

12 11 25 37

12, 11, 25, 37, 52, 92

Ph: 844-844-0102

4th Iteration.

$\overset{11}{\cancel{12}}, \overset{12}{\cancel{11}}, 25, 37, 52, 92$

11, 12, 25, 37, 52, 92.

5th Iteration

It remains unchanged as it is already sorted

11, 12, 25, 37, 52, 92.

---

# Topic Quick Sort

## 13.1 Why do we need another sorting algorithm?

| | Time Complexity | Space Complexity |
|---|---|---|
| 1. Insertion Sort | $O(n^2)$ (worst)<br>$O(n)$ (best) | $O(1)$ |
| 2. Merge Sort | $O(n \log n)$ | $O(n)$ |
| 3. Bubble Sort | $O(n^2)$ (worst)<br>$O(n)$ (best) | $O(1)$. |

→ The best algorithm so far is Merge Sort as it takes $O(n \log n)$ time but it takes $O(n)$ space which is additional space.

→ Can we have another $O(n \log n)$ with better space complexity?

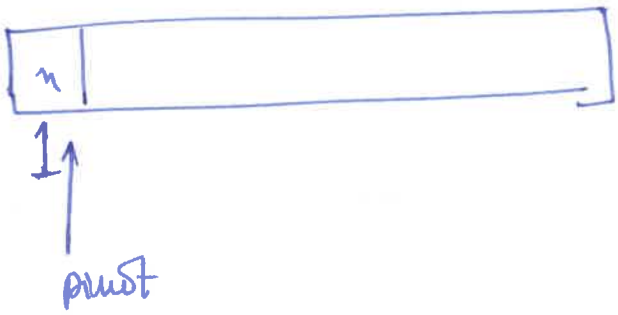
The Answer is Quick Sort.



Mail: Gatecse@appliedcourse.com

## 13.2 How It Works : Intuition

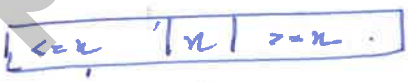
— Also in Quick sort we use Divide and Conquer :



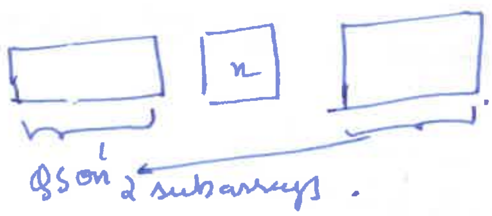

— We select one element as the pivot.

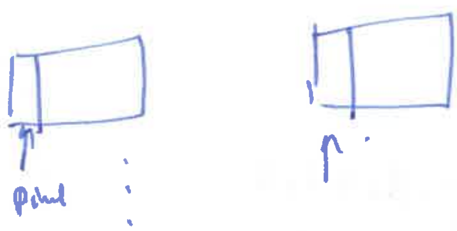
— We have

1. Divide Step : At the end of this step our array will be divided into 2 partitions one containing all elements less than the pivot and the second containing all elements greater than the pivot, at the end of this step the pivot will be placed at its correct position.




(2) Conquer :- Recursively sort the two sub arrays (partitions).

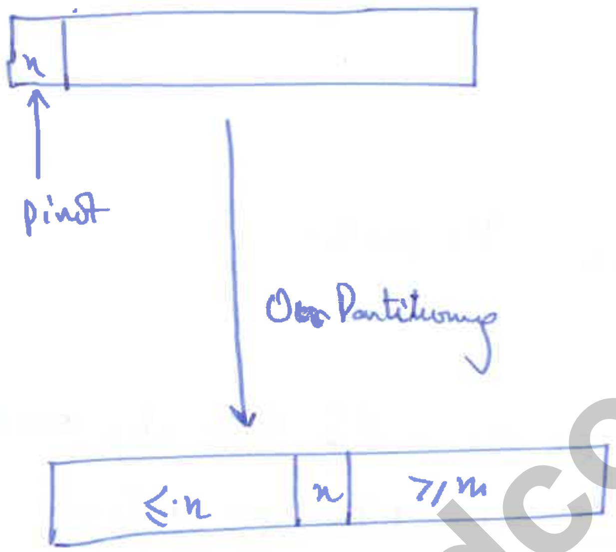


→ The key or important part is that of the partition array procedure.

---

## 13.3 PARTITIONING.




Example of partitioning

| 6, 10, 13, 5, 8, 3, 2, 11
↑ i ↑ j

Here 1st element has been choosen as the pivot.

pivot = 6.

- The complete array is iterated and compared.

6, 10, 13, 5, 8, 3, 2, 11
↑ i ↑ j

6, 5, 13, 10, 8, 3, 2, 11.
↑ i ↑ j

→ Note: All elements from the $i+1$ th position to $j$th position are $\geq$ pivot (6.)

All elements from 2nd to $i$th position are $\leq$ pivot (6).

All elements from $j+1$ to $n$th are not been processed.

6, 5, 13, 10, 8, 3, 2, 11

↑ $i$ ↑ $j$

6, 5, 13, 10, 8, 3, 2, 11

↑ $i$ ↑ $j$

6, 5, 3, 10, 8, 13, 2, 11

↑ $i$ ↑ $j$

6, 5, 3, 2, 8, 13, 10, 11

↑ $i$ ↑ $j$

6, 5, 3, 2, 8, 13, 10, 11

↑ $i$ ↑ $j$

As in the above example 2 to $i$th location we do not have any element $> 6$ all are $\leq 6$. and from $i+1$ to $j$ all a $> 6$

Finally the first is swapped with the $i$th element

2, 5, 3, 2, 6, 13, 10, 11

$\leq 6$ ↑ pivot $> 6$.

PARTITION (A, p, r)

A[p ... q].
pivot = A[p]

```
x ← A[p]
i ← p.
for j ← p+1 to q.
    do if A[j] <= x
        then i ← i+1
             exchange A[i] ←→ A[j].
exchange A[p] ←→ A[i].
return i.
```

Invariant

| x | <= x | >= x | |
|---|---|---|---|
| p | p+1 ... i | j | q |

- Run time for the PARTITION algo is $O(n)$

## 13.4 QUICK SORT BY RECURSION

- The partition function takes an array of size n and selects one element as the pivot and divides the array into 2 parts such that one is <= the pivot and the other is > the pivot.

→ Now We have two unsorted sub arrays which have to be sorted

```
QUICK-SORT (A, p, r)
    if p<r
        then q ← PARTITION (A, p, r)
             QUICKSORT (A, p, q-1) // on left sub Array
             QUICKSORT (A, q+1, r). // on right sub array.
```

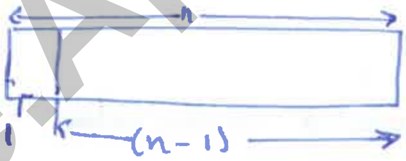
Initial Call : QuickSort (A, 1, n).

---

## 16.5 Time Complexity: Best And Worst Cases

- A problem of size n is divided into 2 problems of size $n_1$ and $n_2$ such that $n_1 + n_2 + 1 = n$, $n_1 + n_2 \simeq n$.

- Worst case




If the pivot selected is either the smallest or the largest part of the array.

$$T(n) = T(0) + T(n-1) + c(n)$$

$$T(n) = T(n-1) + c(n)$$

$$T(n-2) + c(n-1)$$

$$T(n-3) + c(n-2)$$

$$+ c2$$

$$\Rightarrow T(n) = c(n + (n-1) + (n-2) + \dots + 1)$$

$$= c\left(\frac{n(n+1)}{2}\right) = \underline{O(n^2)}$$

- If we are selecting the 1st item as the pivot then the worst case is encountered if the array is already sorted in ascending/decreasing order.

Best Case:-

Lets say that the pivot divides the array into 2 halves then the complexity in such case.

$$T(n) = T(n_1) + T(n_2) + cn.$$

$$n_1 + n_2 + 1 = n.$$

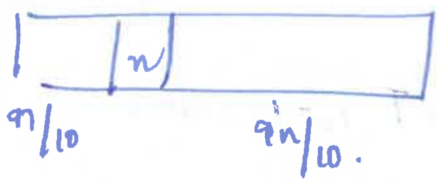
$$n_1 + n_2 = n.$$

$$n_1 = n_2 \simeq n/2.$$

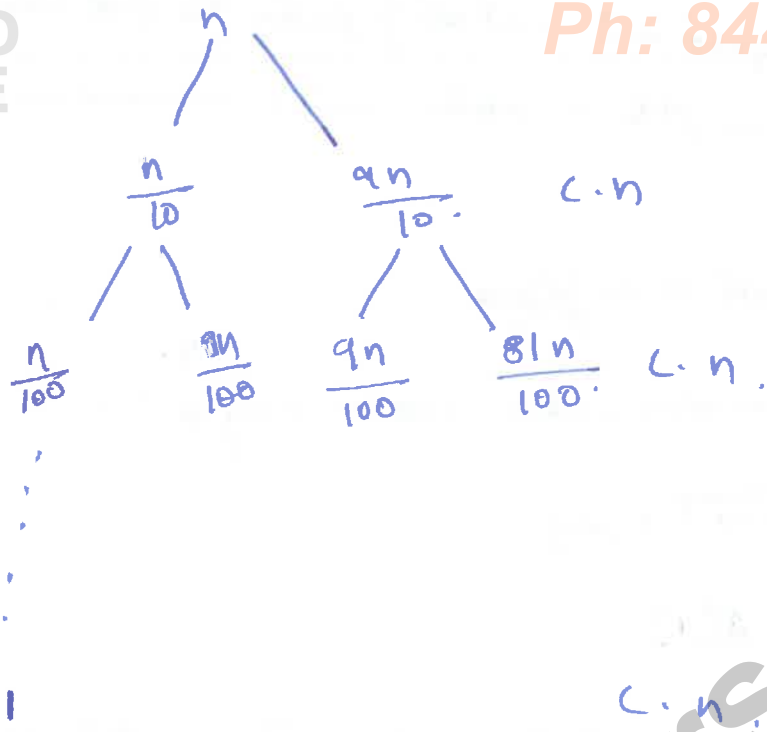
$$T(n) = 2T(n/2) + cn.$$ (same as merge sort recurrence relation).

$$T(n) = O(n \log n).$$

Almost best case: If the array is divided in the ratio 1:9.




$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + cn.$$




The height of the tree is given by $\log_{9/10} n$.

At every level the time taken $O(n)$

$\therefore$ Total time complexity $= O(n \log n)$

Even if we have 90% and 10% split is there we have $O(n \log n)$. time complexity.

## 13.6 Randomized Quicksort And Amortized Analysis

We know when we have Quick Sort in the worst case $O(n^2)$.

$T(n) = T(n-1) + cn$

- Can we fix it? —

Randomized Quick Sort !!!

— In randomized quick sort instead of picking the first element or final element we pick a random number between 1 to n as the pivot.

The steps would be as follows.

1. pick a random number 1 to n. say m

2. $A[1] \xrightarrow{swap} A[m]$

3. pivot = A[1].

This way we can avoid the worst case of a sorted array. we can prove mathematically that $T(n) = O(n \log n)$.

---

## 13.7 Solved Problem Gate 2016.

Q) Assume the algorithms considered here sort the input sequence in ascending order. If the input is already in ascending order, which one of the following is TRUE?

I. Quick sort runs in $\theta(n^2)$ time
II. Bubble sort runs in $\theta(n^2)$ time
III. Merge sort runs in $\theta(n)$ time
IV. Insertion sort runs in $\theta(n)$ time

A. I and II only B. I and III only C. II and IV only D. I and IV only.

Ans

1. I is true. as we know.

2. Bubble sort runs in $O(n)$ time so II is false.

3. Merge sort takes $O(n \log n)$ in all the cases III is false.

4. Insertion sort will takes $\theta(n)$ time in case of sorted algorithms
   this is true

Option D is true.

---

## 13.8 Solved Problem GATE 2009

Q) In quick sort, for sorting n elements, the $(n/4)$th smallest element is selected as pivot using an $O(n)$ time algorithm. What is the worst case time complexity of this Quicksort algorithm

1. $\theta(n)$
2. $\theta(n \log n)$
3. $\theta(n^2)$
4. $\theta(n^2 \log n)$

$$T(n) = \theta(n) + O(n) + T\left(\frac{n}{4}\right) + T\left(n - \frac{n}{4}\right)$$

$$T(n) = O(n) + T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right)$$

we know it is $\theta(n \log n)$ option 2.

# 13.9 Solved Problem GATE 2001



Q) Randomized QS is an extension to Quick Sort where the pivot is choosen randomly. What is the worst case complexity of sorting n elements using Randomized Quick sort.

1. $O(n)$
2. $O(n \times \log n)$
3. $O(n^2)$
4. $O(n!)$

Ans

If all the elements are the same.

3, 3, 3, 3, 3, 3.

In such a case whichever pivot is choosen. the pivot will be set to the first or last place.

3, 3, 3, 3, 3, 3.

or

3, 3, 3, 3, 3, 3

$$T(n) = T(n-1) + cn$$

$$= O(n^2).$$

Also even in the case of randomized picking of the pivot there is a very less probability that we choose the smallest element at every pass, even though such a case is not much probable but there is a chance that it will arise in such a case also randomized quick sort takes $\Theta(n^2)$ time. option 3. is correct.

---

## 13.10 Solved Problem Gate 2008

Q) Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sublists each of which contains at least one-fifth of the elements. Let $T(n)$ be the number of comparisons required to sort $n$ elements. Then.

1. $T(n) \le 2T(n/5) + n$
2. $T(n) \le 2T(n/5) + T(4n/5) + n$
3. $T(n) \le 2T(4n/5) + n$.
4. $T(n) \le 2T(n/2) + n$.

Option 2 is the most appropriate one.

## 13.11 Solved Problem GATE 2015

Q) Which of the following recurrence relations for the worst case time complexity of the quick sort algorithm for sorting $n (\geq 2)$ number? In the recurrence equations given in the equation below, c is a constant.

1. $T(n) = 2T(n/2) + n$
2. $T(n) = T(n-1) + T(1) + n$
3. $T(n) = 2T(n-1) + n$
4. $T(n) = T(n/2) + n$

We know that in the worst case QS breaks the array into two parts of (1) and (n-1) elements, the option 2 is the correct option.

## 13.12 Solved Problem GATE 2014

Q) You have an array of n elements. Suppose you implement quick sort in such a way that you are choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is

1. $O(n^2)$
2. $O(n \log n)$
3. $\Theta(n \log n)$
4. $O(n^3)$

— Which ever element is choosen for the pivot, the worst case is when we have the smallest/largest is choosen as the pivot.

the worst case time complexity = $\underline{O(n^2)}$ option 1.

## 13.13 Solved Problem GATE 2019

Q) An array of 25 distinct elements is to be sorted using quick sort. Assuming that the pivot element is choosen randomly. The probability that the pivot element gets placed in the worst possible location in the partitioning (rounded off to 2 decimal places) is ___


The worst case would arise when we pick either the minimum or maximum elements.

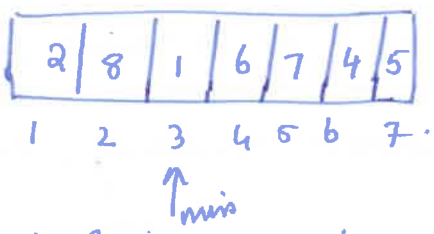
∴ As all the elements are unique, there exists a unique minima and a unique maxima.

$$\therefore \; P(\text{Worst case}) = \frac{2}{25} = \underline{0.08}$$

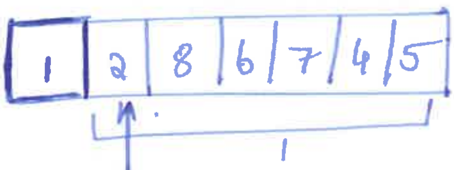
# Topic Selection Sort

## 15.1 How It Works: Intuition + Code

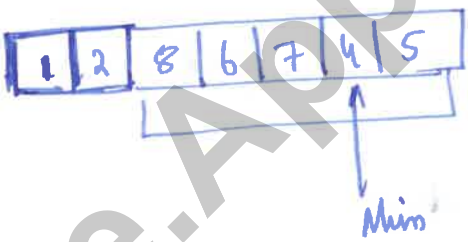
Let us consider the following array

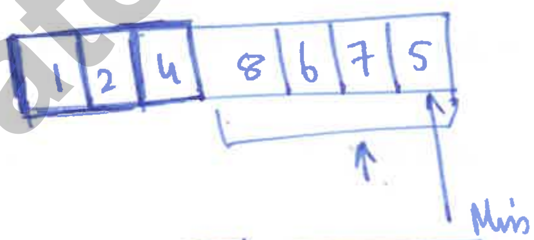


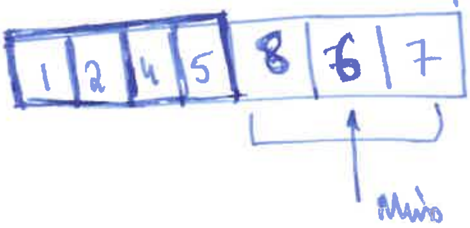

first pass find the min element.




Min smallest of the unsorted subarray.




| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 6 | 7 | 8 |

Final Sorted array is 1, 2, 4, 5, 6, 7, 8.

→ At each step we find the minimum sub array which is unsorted and then we place it at the end of the sorted sub array.

## Implementation

```
int i,j;
int n;
for(j=0; j<n-1 ; j++)
 {
    // find the min element for the array a[j ---- n-1]
    // assume min is first elmt .
    int iMin =j;
    for( i=j+1 ; i<n; i++)
     {
        iMin = i;
     }
 }
 if (iMin !=j)
  {
     swap( a[j] , a[iMin] );
  }
}
```

# 16.2. Space And Time Complexity





- The outer loop runs $n$ times

- When $j=0$: inner loop makes $(n-1)$ comparisons + 1 swap.

  when $j=1$ " $(n-2)$ " + 1 swaps.

  $j=2$ " $(n-3)$ " + 1 swap.

  ⋮

  $j=n-1$ " $(1)$ comparison + 1 swap.

$\therefore$ Total # of comparisons $= 1+2+3+\cdots(n-1)$

$= O(n^2)$ comparisons.

and $O(n)$ swaps.

- Best and Worst case we have $O(n^2)$ time complexity.

- Space complexity $O(1)$ as it is an inplace sorting algorithm.

- For selection sort we have only $O(n)$ swaps in the worst case whereas in insertion sort we have $O(n^2)$ swaps.

# 15.3 When To Use Selection Sort



- Where is selection sort more useful?

  - For small arrays (0-20) elements we can use any insertion or selection sort

  - But for large arrays we need to avoid both Insertion & Selection sort as $O(n^2)$ grows fast.

- Why do we need to know selection sort when we have insertion sort already?

  Swap corresponds to write } - in the RAM
  comparison corresponds to read

  in the RAM chips $T_{write} \gg T_{read}$ when we have more difference b/w $T_{write}$ and $T_{read}$ then Selection sort should be preferred in such cases.

## 15.4 Solved Problem GATE 2010, 2009



Q) Which of the following is the tightest upper bound that represents the number of swaps required to sort n numbers using selection sort

1. $O(\log n)$
2. $O(n)$
3. $O(n \log n)$
4. $O(n^2)$

Ans 2. As we have discussed an known # of swaps $O(n)$.

## 15.5 Solved Problem GATE 2007

Q) Which of the following sorting algorithms has the lowest worst case complexity?

1. Merge Sort.
2. Bubble sort
3. Quick Sort.
4. Selection Sort.

Ans
1) M — $O(n \log n)$ ✓
2) B — $O(n^2)$
3) Q — $O(n^2)$
4) S — $O(n^2)$

# 16.6 Solved Problem 2016.



The worst case running time of Insertion sort, Merge sort and Quick sort respectively are?

(A) $\theta(n \log n)$, $\theta(n \log n)$ and $\theta(n^2)$

(B) $\theta(n^2)$, $\theta(n^2)$ and $\theta(n \log n)$

(C) $\theta(n^2)$, $\theta(n \log n)$ and $\theta(n \log n)$

(D) $\theta(n^2)$, $\theta(n \log n)$ and $\theta(n^2)$

Ans Option D as we know the worst case time complexity for IS $\theta(n^2)$

· MS $\theta(n \log n)$

· QS - $\theta(n^2)$



# 16.7 Sample Questions

Q) Which of the following sorting algorithms has a running time that is least dependent on the initial ordering on the inputs?

1. Quick Sort.
2. Insertion Sort
3. Merge Sort.
4. Selection Sort.

Ans.

QS. 1. If the array is already sorted then QS will take $\theta(n^2)$. Otherwise it is $\theta(n \log n)$ it is not possible.

2. Insertion Sort :- Here we have $O(n)$ for sorted
$O(n^2)$ for worst case sorted in the opposite order.

3. Merge Sort :- $O(n \log n)$ always ~~so~~. it is a possible answer.

4. Selection Sort :- $O(n^2)$ in the best and worst case
but here in it is already sorted we have
$O(1)$ swaps and $O(n^2)$ comparisons as swaps take more time
than comparisons therefore there is a diffirence
in the run time (in the question the run time
is asked for).

$\therefore$ The least dependent on ordering of i/p Merge Sort option 3.

# Topic Linear Time Sorting

## 16.1 Lower Bounds On Worst Case Of Comparision Sorting

- All the sorting algorithms we have discussed we have different sorting. The worst case time complexity is either $O(n \log n)$ or $O(n^2)$
- Is is possible to have a sorting algorithm with $O(n)$ worst case?
- All of the above sorting algorithms performing comparisions among themselves, there are known as comparision based sorting algorith.

Q Can there be any comparision based sorting algo with worst case $O(n)$?

Its not possible following is the proof.

Lets consider the array $A = [a_1, a_2, \ldots, a_n]$

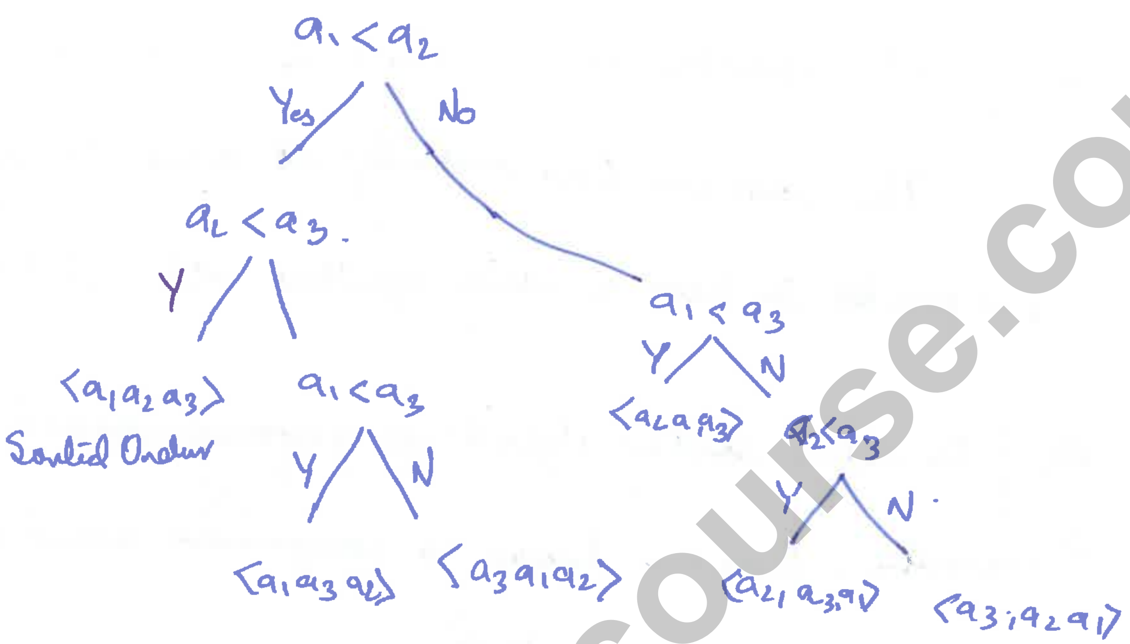
$n = 3$ such that $a_i \neq a_j$ $\begin{cases} a_i > a_j \\ a_i < a_j \end{cases}$

$\langle a_1, a_2, a_3 \rangle$.

given 3 ways we can arrange them in 3! way.

$a_1\ a_2\ a_3$
$a_1\ a_3\ a_2$ → If we have n numbers in n! ways.
$a_2\ a_1\ a_3$
$a_2\ a_3\ a_1$
$a_3\ a_1\ a_2$
$a_3\ a_2\ a_1$

If we have three 3 numbers and we want to compare them lets construct a decision tree for it, known as Binary Decision Tree.




Lets take an example $a_1 = 6$ $a_2 = 4$ $a_3 = 7$

- If we follow the path from the root down we will land at the root node $\langle a_1, a_3, a_2 \rangle$ which is the correct sorted order.

- If we note we have $6 = 3!$ leaf nodes which are the all the possible arrangements of the 3 numbers.

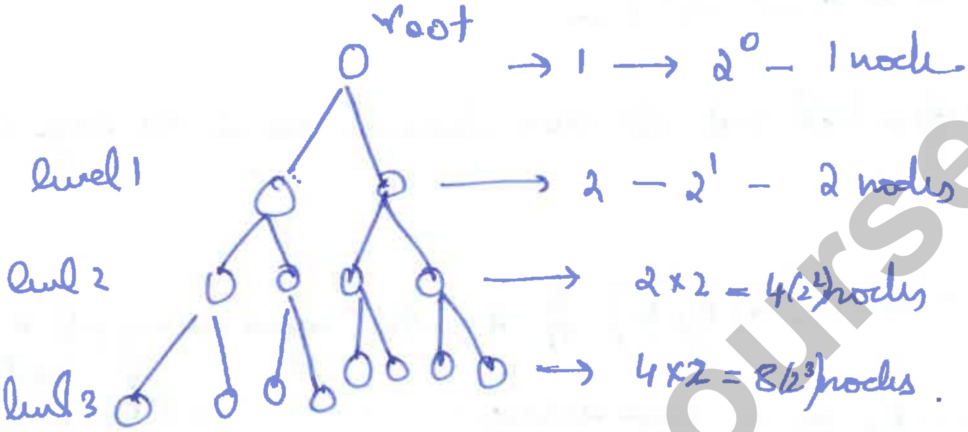
  $\#$ leaf nodes $= n!$

- The Binary Decision tree is an abstract mathematical model to sort the elements using comparisions.

- Instead of studying many algorithms we can anlyze the binary decision tree.

— When we sort an array using the binary decision tree the maximum number of comparisions we need to make is given by the hight of the tree as the each node represents a comparision and only one node is traversed at a particular level.

Binary Decision tree



- A binary tree of ht h can have at most $2^h$ nodes on leaf nodes

- We know that we have $n!$ leaf nodes in our case.

$\therefore$ Max possible hight $2^h \geq n!$

$h \geq \log n!$

$h = \Omega(n \log n)$ (as $\log(n!) = \Omega(n \log n)$)

$\therefore$ # of comparisions we need to make $= \Omega(n \log n)$

$\therefore$ As long as we are using a comparision based sorting algorithm # comparisions $= \underline{\underline{\Omega(n \log n)}}$

# 16.2 Counting Sort : Intution + Code



— Non-comparision based sorting algorithm.

A:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

Here we know before hand that all our elements are in the range 0 to 5 → k.

COUNTING-SORT(A, B, k) // A is the input array, B is the o/p sorted array and k is the range of elements.

1. let C[0...k] be a new array.
2. for i = 0 to k
3. C[i] = 0.
4. for j = 1 to A.length
5. C[A[j]] = C[A[j]] + 1
6. // C now contains the number of elements equal to i.
7. for i = 1 to k.
8. C[i] = C[i] + C[i − 1]
9. // C[i] now contains the number of elements less than or equal to i.
10. for j = A.length down to 1
11. B[C[A[j]]] = A[j]
12. C[A[j]] = C[A[j]] − 1

→ line 1 creates an auxiliary array of size k.

→ lines 2-3 are initializing all elements of C as 0.

→ lines 4-5 we are counting occurrence of every number in between 0 to k and recording at the corresponding index in array C.

→ The loop in lines 7-9 store the cumulative sum in the array C upto element i in the ith location in other words it now contains the number of elements less than or equal to i.

→ lines 10-12 form the main part, Here element from A is choosen and placed at the C[A[j]] th location as C[A[j]] represents the number of items / elements which are less than or equal to A[j] then we can place it at C[A[j]] th location and then we are decrementing the value of C[A[j]] as one element has been placed.

→ Let us consider the example

A

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

B

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

$k = 5$

→ After line 2-3 we have array C.

C

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |

- After the lines 6-8 we are left with

| | | | | | | |
|---|---|---|---|---|---|---|
| C | 2 | 0 | 2 | 3 | 0 | 1 |
| | 0 | 1 | 2 | 3 | 4 | 5 |

- After the lines 7-8 we have the cumulative size array.

| | | | | | | |
|---|---|---|---|---|---|---|
| C | 2 | 2 | 4 | 7 | 7 | 8 |
| | 0 | 1 | 2 | 3 | 4 | 5 |

- In the loop from lines 10-12 Array B is constructed using array C.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| B | 0 | 0 | 2 | 2 | 3 | 3 | 3 | 5 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | |
|---|---|---|---|---|---|---|
| C | ~~2~~ ~~1~~ 0 | 2 | ~~4~~ 3 | ~~7~~ ~~6~~ 5 | 7 | ~~8~~ 7 |
| | 0 | 1 | 2 | 3 | 4 | 5 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A | 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

→ $i = 8$

$A[8] = 3$

$C[A[8]] = 7$

at location 7 A[8] is placed in array B.

→ $i = 7$

$A[7] = 0$

$C[A[7]] = 2$

At 2 A[7] is placed

→ $i=6$

$A[6] = 3$

$C[A[6]] = 6.$

At 6 $A[6]$ is placed.

→ $i=5$

$A[5]=2.$

$C[A[5]] = 4$

At 4 $A[5]$ is placed.

→ $i=4$

$A[4] = 0.$

$C[A[4]] = 1$

At location 1 $A[4]$ is placed.

→ $i=3$

$A[3] = 3$

$C[A[3]] = 5$

At location 5 $A[3]$ is placed.

→ $i=2.$

$A[2] = 5$

$C[A[2]] = 8.$

At 8 $A[2]$ is placed.

→ $i=1$ $A[1] = 2$

$C[A[1]] = 3$

At location 3 $A[1]$ is placed.

- Counting sort ensures stability i.e. also A[6] and A[8] are the same say = 3 but then also in the sorted array their order is preserved. A[6] is moved to B[6] and A[9] to B[8].

---

## 16.3 Space And Time Complexity

Time Complexity.

1. Lines 2-3 take $O(k)$ time
2. Lines 4-5 take $O(n)$ time
3. Lines 7-8 take $O(k)$ time
4. Lines 10-12 take $O(n)$ time

Total time complexity = $O(n+k)$.

If k is $O(n)$ then. the time complexity reduces to $O(n)$.

Space Complexity.

A is of size $-n$.

- If B is given as I/p. then we need not consider it but if it is not given it is $O(n)$ size.
- Array C is $O(k)$ size.

If B is given as I/p then space complexity $O(K)$ otherwise it is $O(n+k)$

## 16.4 Radix Sort

| | | | |
|---|---|---|---|
| 329 | 720 | 720 | 329 |
| 457 | 355 | 329 | 355 |
| 657 | 436 | 436 | 436 |
| 839 | 457 | 839 | 457 |
| 436 | 657 | 355 | 657 |
| 720 | 329 | 457 | 720 |
| 355 | 839 | 657 | 839 |

- When we are sorting using counting sort on the units place the remaining 2 digits are considered as satellite data, similarly for the other 2 cases, the digits apart from the currently sorted digit are considered as satellite data.

→ Counting sort is a stable sorting algorithm, if it is not stable then Radix sort will not work correctly.

### Time Complexity.

If we have n elements and each is of d digits then we have to apply counting sort d no of times. Time complexity $= O(n \times d)$

- In case of numbers k is a constant $k=9$ for decimal.

## 16.5 Where to Use Which Sorting Algorithm

→ Insertion sort is preferable when we have an almost sorted array. It takes $O(n)$ time complexity

→ Merge Sort :- When our array does not fit in our memory.

→ Quick Sort :- Very good general purpose sorting algorithm.
(with randomization)

→ Counting & Radix Sort } - If range is known and we have multiple digits we can get linear time algorithm.

→ Bubble Sort :- Should be avoided.

---

## 16.6 Solved Problem GATE 2008

Q) If we use Radix Sort to sort n integers which are in the range $(n^{k/2}, n^k]$, for some $k > 0$, which is independent of n, the time taken would be?

1. $\Theta(n)$
2. $\Theta(k \times n)$
3. $\Theta(n \times \log n)$
4. $\Theta(n^2)$

Ans

Radix sort time complexity $O(an)$
$\downarrow$
# of digits.

No of digits to represent a number in a system of basis $b = \lceil \log_b n \rceil$

for example for 4 numbers in binary we need $\lceil \log_2 4 \rceil = 2$ digits

| | |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

similarly for 16 numbers we need $\lceil \log_2 16 \rceil = 4$ digits.

$\therefore$ to represent the number $n^k$ we need $\log n^k$.

$= k \log n$ digits

and we have $n$ such "$k \log n$" digit numbers.

$\therefore$ The time complexity $O(n \times k \log n)$ as $k$ is considered as a constant $O(n \log n)$.

# Topic Linear Search

## Video 35.1 Linear Search : Intuition And Code .

- Searching : The search for a given key/element in an unsorted array we need to check if it is present or not.

```
LinearSearch(a, x)
{
    for i=1 to a.length
    {
        if a[i] = x
            return "found @ i"
    }
    return not found
}
```



- Time Complexity :- $\Theta(n)$ as we compare the complete array to search until it is found.

- Space Complexity :- $\Theta(1)$ because it takes no additional space.

- There could be a variation to linear search where we want all occurrences. If the key even is found in that case time complexity $\Theta(n)$ and space complexity $\Theta(1)$, the loop should run till the end.

# Topic Binary Search

## 36.1 Intuition

- From linear search we know that time complexity $O(n)$ for an unsorted array. Can we do better?

. The answer is Binary Search!

- In Binary Search we have i/p array which is sorted and a given key. we can search for the key in $O(\log n)$ time.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 8 | 10 | 12 | 14 | 16 | 19 |



**Binary Search.**

we calculate the middle element $m = \left\lfloor \frac{l+r}{2} \right\rfloor$

if $x == A[m] \longrightarrow$ Found $x$ in A.

$x > A[m] \longrightarrow l = m+1,\ r = r$

$x < A[m] \longrightarrow l = l$
$r = m-1$

Lets consider the above example key $= 6$.

(1) $l=1 \quad r=10$

$$m = \left\lfloor \frac{r+l}{2} \right\rfloor = \left\lfloor \frac{11}{2} \right\rfloor = 5$$

$x = 6 < A[m] = 8.$

(2) $l=1 \quad r = m-1 = 5-1 = 4$

$$m = \left\lfloor \frac{r+l}{2} \right\rfloor = 2$$

$x = 6 \quad A[m] = 4$

$x > A[m]$

(3) $l = 3 \quad r = 4$

$$m = \left\lfloor \frac{3+4}{2} \right\rfloor = 3$$

$A[m] = 6.$

$x == A[m]$

FOUND X in A at location 3

→ If $x = 7$ on the same array.

$x > A[m] = 6$.

$l = 4$

$r = 4$

(4) $l = 4$ $r = 4$ $m = 4$.

$A[m] = 8 > x = 7$.

$r = m - 1 = 3$

$l = 4$.

$\underline{r < l}$. stop!!

The range in between $l$ to $r$ is the current search space always.

⇒ In binary search in every iteration we are shrinking the search space by $1/2$.


## 36.2. Pseudo Code

— For binary search we have Iterative and recursive version for Binary search which perform the same logic.

```
BINARY-SEARCH (A, x, l, n)
{
    l = 1 ; r = n
    while (l ≤ n)
    {
        m = ⌊(l+r)/2⌋ ;

        if x < A[m]
            r = m-1

        if x > A[m]
            l = m+1

        if x == A[m]
            return "FOUND"
    }.

    return "NOT FOUND"
}
```



Time Complexity

At Each step we are reducing a problem of size $n$ to $n/2$

$n \downarrow n/2 \downarrow n/4 \; \vdots \; 1$

$T(n) = T(n/2) + c$.

we know it can be solved in $\theta(\log n)$ time.

$\underline{T(n) = \theta(\log n)}$

For the iterative version we have space complexity $\underline{\theta(1)}$

```
BinSearchRecursive ( A, x, l, r)
{
    if (l > r)
    {
        return "NOT FOUND"
    }

    m = ⌊(l+r)/2⌋

    if x = A[m]
        return "FOUND".

    if x < A[m]
        BinSearchRecursive(A, x, l, m-1)

    if x > A[m]
        BinSearchRecursive(A, x, m+1, r)
}
```

* Binary Search works only on sorted arrays.
* Binary Search does not work in case of a linked list (if we use it then we cannot get $\theta(\log n)$ time)

# 46.8 Comparision Of All Sorting Methods

Reference link :- https://en.wikipedia.org/wiki/Sorting_algorithms.

| | Name | Best | Avg | Worst | Memory | Stable |
|---|---|---|---|---|---|---|
| Comparision Based. | | | | | | |
| 1. | QuickSort | $n\log n$ | $n\log n$ | $n^2$ | $\log n$ | Not Stable |
| 2. | Merge Sort | $n\log n$ | $n\log n$ | $n\log n$ | 1 | Stable |
| 3. | Heap Sort | $n\log n$. if all keys are distinct | $n\log n$ | $n\log n$ | 1 | Not Stable |
| 4. | Insertion Sort | $n$ | $n^2$ | $n^2$ | 1 | Not Stable |
| 5. | Selection Sort | $n^2$ | $n^2$ | $n^2$ | 1 | Not Stable |
| 6. | Bubble Sort | $n$ | $n^2$ | $n^2$ | 1 | Stable |
| Non comparision Based. | | | | | | |
| 7. | Counting Sort | – | $n+r$ | $n+r$. | $n+r$ | Stable |
| 8. | Radix Sort | $O(n \times d)$ | $O(n \times d)$ | $O(n \times d)$ | 1 | Stable |

More sorting algorithms can be found on the reference page.

Q) Which of the following inplace sorting algorithm needs the minimum number of swaps?

A. Quick Sort

B. Insertion sort

C. Selection Sort

D. Heap sort.

Ans

In Quick sort we need $O(n^2)$ swaps.

in insertion sort also we get $O(n^2)$ swaps.

in selection sort in worst case we get $O(n)$ swaps

in Heap sort in the worst case we get $O(n \log n)$ swaps.

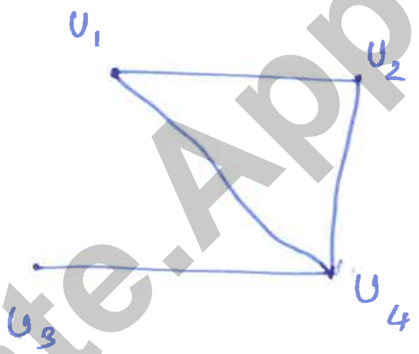
$\therefore$ Min is option C Selection Sort

# TOPIC 60 GRAPHS - 1

## 60.1 Graphs : Why, What and Basis

- A Graph is a type of data structure.
- A Graph is represented by a set of vertices and edges.
  $G=(V,E)$.
- A tree is also a type of a graph.
- Graphs also have many applications in the real world.
  For example in Facebook.

  If $U_1, U_2, U_3$ and $U_4$ are users they can be represented as vertices and the edges represent the friendship among the users




$E = \{ (U_1, U_2), (U_2, U_4), (U_3, U_4), (U_1, U_4) \}$.

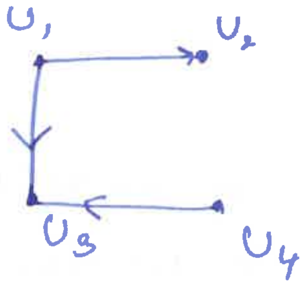
$V = \{ U_1 U_2 U_3 U_4 \}$.

The above is known as friendship graph.

$(U_1 U_2)$ is equivalent to $(U_2, U_1)$ in an undirected graph.

- Fb uses a collection of graph algorithm to recommend friends.

→ Another example is of Twitter / Instagram




← Followers Graph

$G = (V, E)$

$V = \{U_1, U_2, U_3, U_4\}$

$E = \{(U_1, U_2), (U_1, U_3), (U_3, U_2), (U_4, U_3)\}$

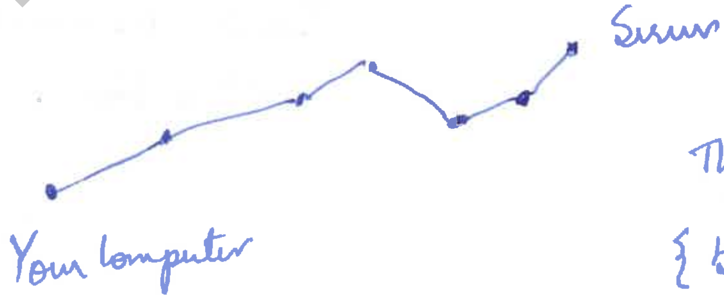
↓

$U_1$ follows $U_2$

$(U_1, U_2) \neq (U_2, U_1)$ – This is an example of a directed graph.

Twitter uses graph algorithms to provide suggestions ~~on what~~ and which users to follow.

→ The internet can also be represented as a graph




This is a very large Graph {billions of Servers}.

There are many Graph Algorithms at work when you browse the internet. (Computer Networks at work).

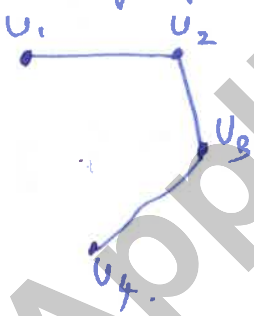
→

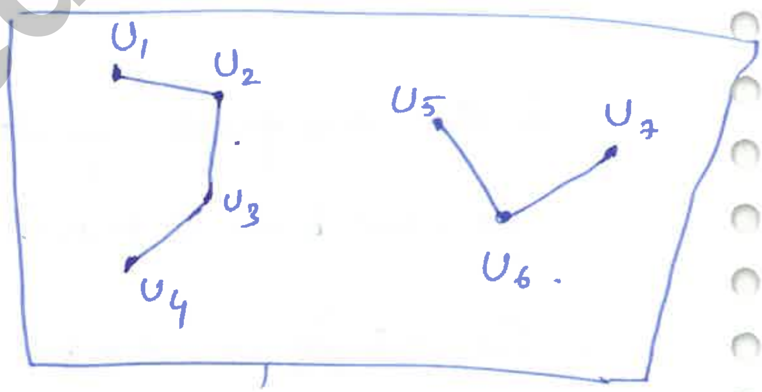
→ In Neuroscience the human brain is modelled as a huge network of neurons.

Billion and billions of neurons and connections among them.

An Artificial Neural Network is one such Network where we try to model human brain by building such a network. This also makes use of a graph.

— A Tree is a special type of graph, it is a connected graph which does not contain any cycles.




This is disconnected and it is not a tree.

Ph: 844-844-0102

Some properties of graphs:

(1) Vertices n- vertices

The maximum number of edges = $\frac{n(n-1)}{2}$.

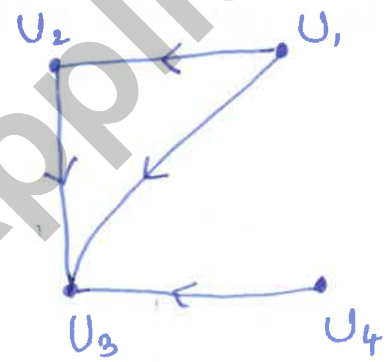
Also we can note that $|E| = O(|V|^2)$. or $\log(|E|) = O(|V|)$

(2) In a connected Graph

$|E| \geq |V| - 1$

## 60.2. Representation of Graphs: Adjacency Matrix

Let us take example of the following directed graph.




$G = (V, E)$

$|V| = 4 = n$

$|E| = m = 4$

- The Adj Matrix is a way to represent a graph in the form of a matrix. we will have to define Adj matrix of.
for a graph with n vertices
n×n size.

- If there exist an edge from $U_i$ to $U_j$ then ith row jth column element is marked as 1 or else it is marked as 0.

Mail: Gatecse@appliedcourse.com

Adjacency list for the above graph is given by

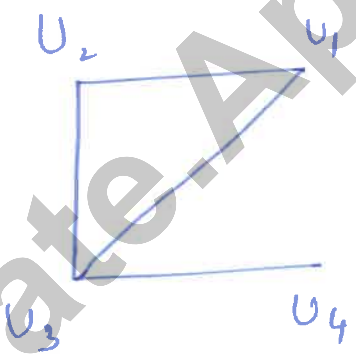
$$
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & & & & \\
2 & & & & \\
3 & & & & \\
4 & & & &
\end{array}
\quad
\begin{bmatrix}
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
$$

$$A_{ij} = A[i,j] = \begin{cases} 1 & \text{iff } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$

An Adj Matrix can be also written for an undirected graph.

for an undirected graph we will have $A_{ij} = A_{ji}$

The adjacency matrix will be a symmetric matrix



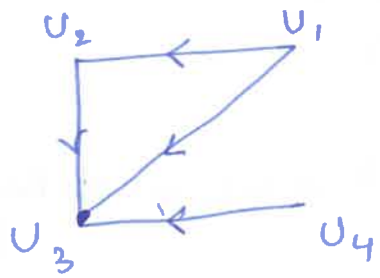

$$
\begin{array}{c|c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\hline
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

→ An Adjacency matrix requires $O(n^2)$ space. ($n$ is the no of vertices it is irrespective of no of edges ($m$))

# 6.3 Representation of Graphs

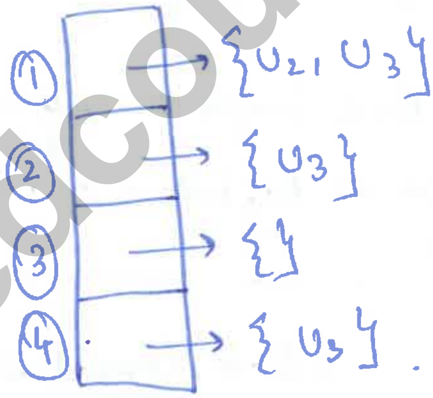
## Adjacency lists

- Another way to represents Graphs other than the adjacency matrix representation




We make use of an array of size n
n = no of vertices

n - vertices
m - edges.




→ Each cell in the array points to the list of neighbouring vertices from that cell / vertex.

→ In the above example $U_1$ has $U_2$ and $U_3$ as neighbours or in other words there is an edge from $U_1$ to $U_2$ and $U_1$ to $U_3$

→ Similarly from $U_2$ to $U_3$ and $U_3$ does not have any outgoing edge.

→ Similarly from $U_4$ there is an edge to $U_3$.

- Each cell in the array points to a linked list, list of neighbouring vertices.



⟶ The space of complexity of the adj. list representation $O(n+m)$

- The space complexity of Adj Matrix $O(n^2)$

- When is Adj list better than Adj matrix:

when $m << n^2$

We know that max no of edges $m = \Theta(n^2)$
and min no of edges $= 0$

When the no of edges $<< n^2$ then in such a case an Adj list is more efficient.

Dense Graph is a graph $m \simeq \frac{n(n-1)}{2}$. Otherwise it is known as a sparse graph.

→ In case of a Dense Graph an Adj matrix is a better choice and in case of a sparse graph an Adj list is a better choice.

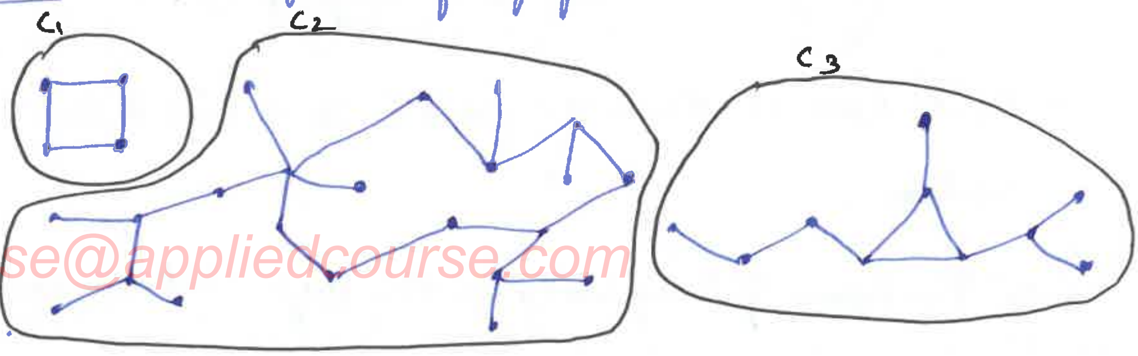
---

## 60.4 Connectivity in undirected Graphs.

<u>Connected Graph</u> :- There exists a path from every vertex to every other vertex $\{v_i \leadsto v_j \ \forall i,j\}$

For example in the internet Graph, connectivity represents if the server is reachable. In a road network we would like to have all the places connected, so that we can reach all the places by means of road.

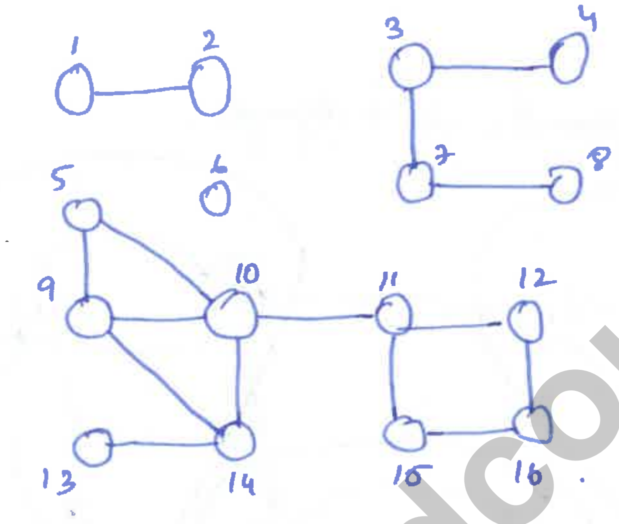
<u>Connected Component</u> :- A component of a graph which is a connected subgraph.




The graph drawn here has 3 connected components $C_1$ $C_2$ $C_3$.

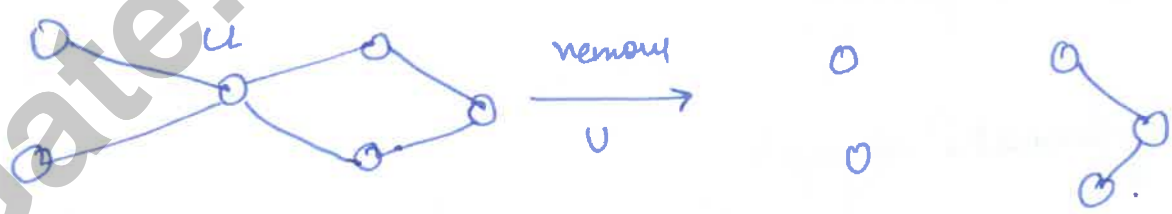
**Bridges / Cut-Edges / Cut arcs** :- A bridge is an edge whose removal results in an increase in the # of connected components of the graph.

In a road network we do not want the bridges to be lost, we loose connectivity if we loose them.

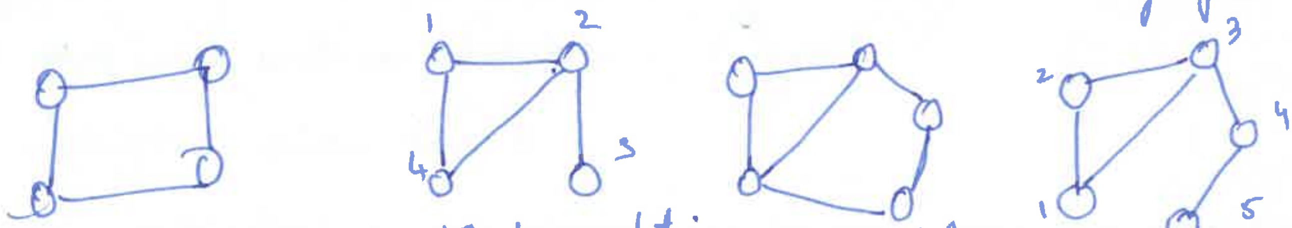



In the above graph the edges (1,2), (3,4), (3,7), (7,8), (13,14), (10,11) are bridges.

**Cut Vertices or Articulation Point** :- An Articulation point is a vertex whose removal will disconnect the graph.



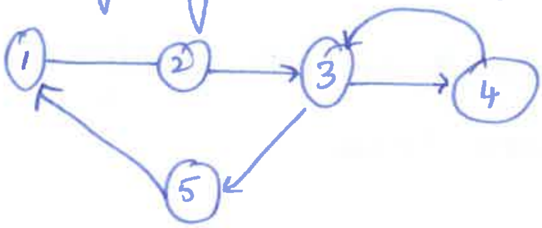

**Biconnected Graph** :- The removal of one vertex does not disconnect the graph.




Biconnected graph. | Not a biconnected because of vertex 2. | Biconnected graph. | Not biconnected because of vertex 4.

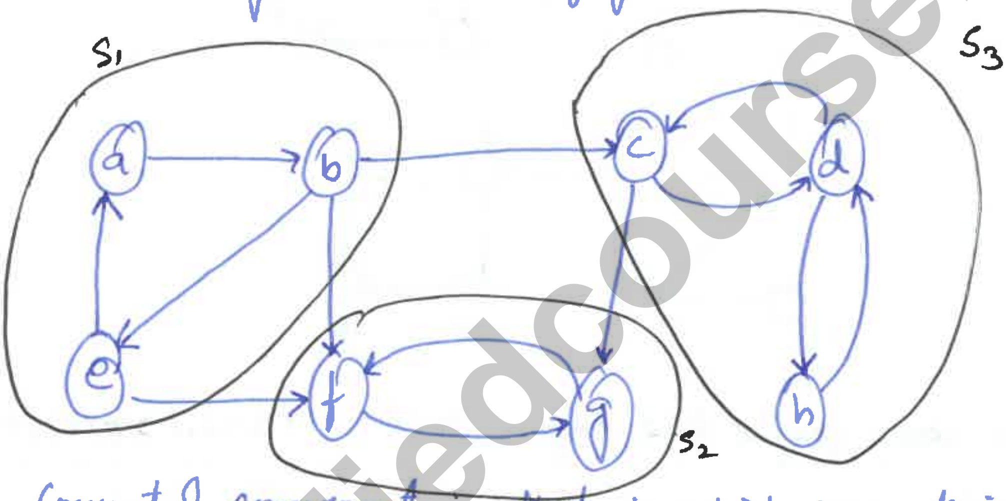
# 60.5 Connectivity in Directed Graphs

1. Strongly Connected digraph :- If there exists a directed path from every vertex $v_i$ to ~~$v_j$~~ any other vertex $v_j$ in G.




~~2. Strongly~~

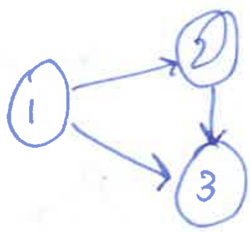
2. Strongly Connected Components in a digraph.




A strongly connected component is that in which every pair of vertices have a directed path ~~to~~ to and from each of them.

→ In the above graph {a,b,c} {f,g} {c,d,h} are strongly connected components.

3. Weakly Connected digraph.

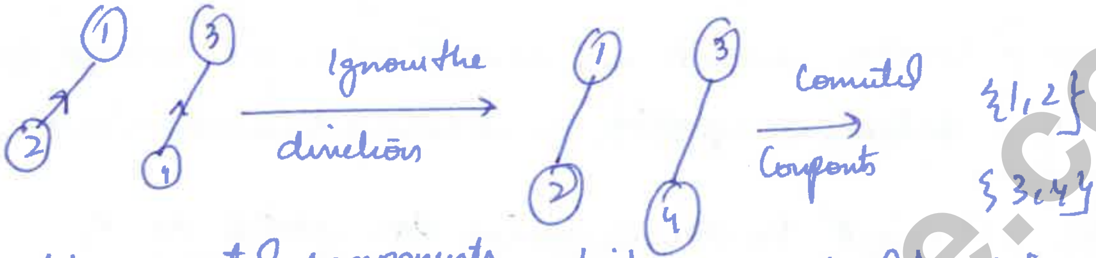



The example shown is not strongly connected as there is no path 3 to 1 which is directed.

But the above digraph if we remove the directions then.


the graph would look like this, which is a connected graph.

## (4) Weakly Connected Components



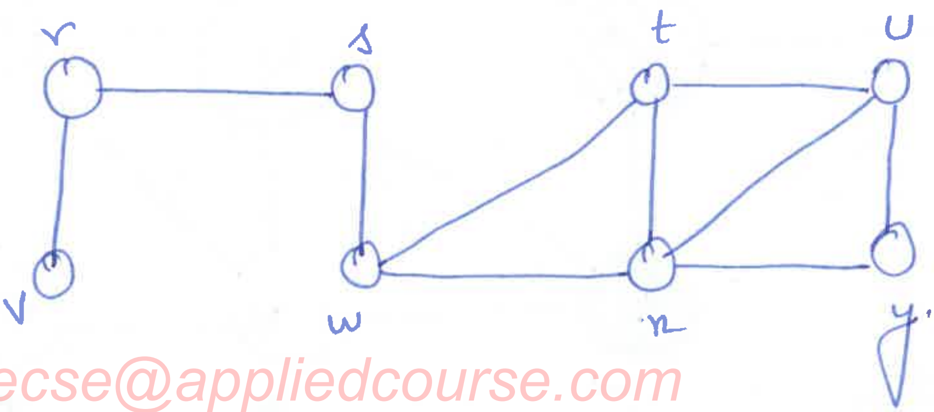

→ The weakly connected components ~~which~~ are obtained by ignoring the directions of the graph edges and then the components which are obtained are the weakly connected components of the digraph.

## 60.6. Breadth First Search: Intuition and Example.

- BFS is a Graph traversal technique
- traversal technique :- A way to visit all the vertices in a particular order based on some strategy

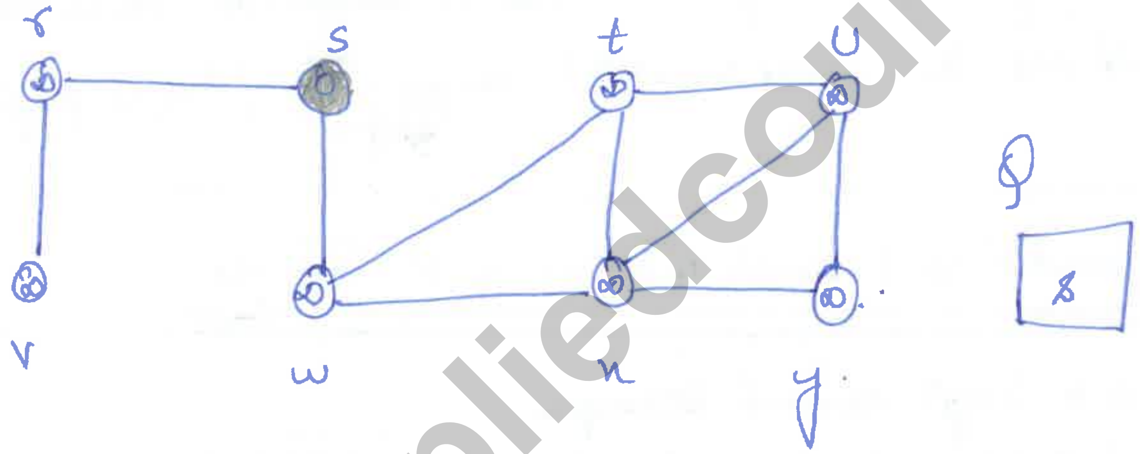
Inputs :- Graph (G, V)
Source vertex s.

Let us consider the following Graph example.



s is the source vertex

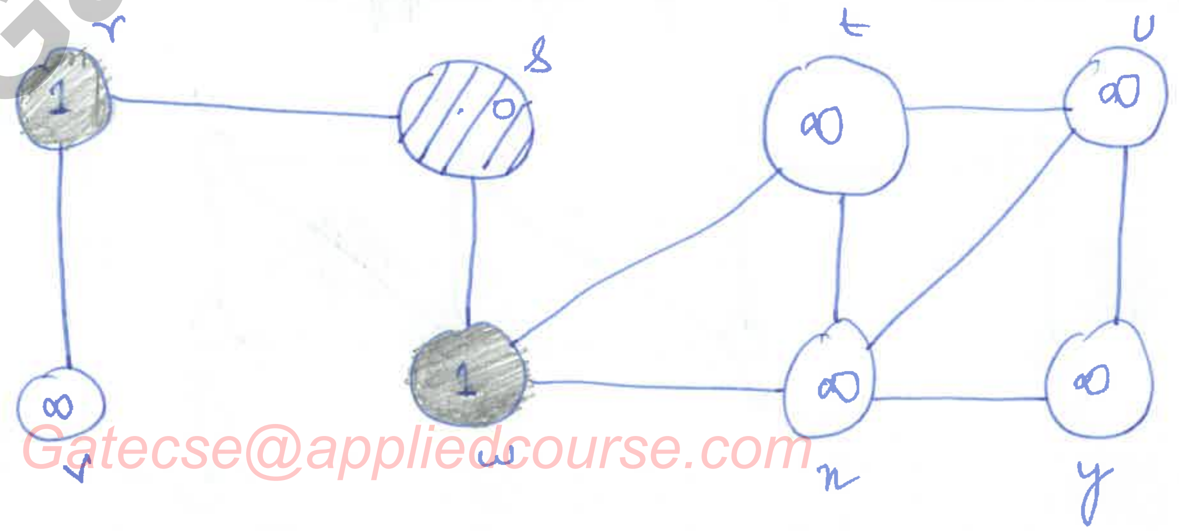
- Each vertex is initially coloured white
- Once it is added to the queue it ~~can be~~ is coloured gray.
- Once it is removed from the queue it is coloured black.
- Once a vertex is dequeued all the neighbouring vertices of the dequeued vertex which are coloured white are coloured gray and their distance is updated as distance of dequeued vertex + 1 and added to the queue.
- Initially all except the source vertex are updated as ∞.
- And the source vertex is added to the queue (Q)




Now s is dequeued and marked black. (represented as ⊘)

Its neighbouring vertices to be added i.e {r, w} to the queue

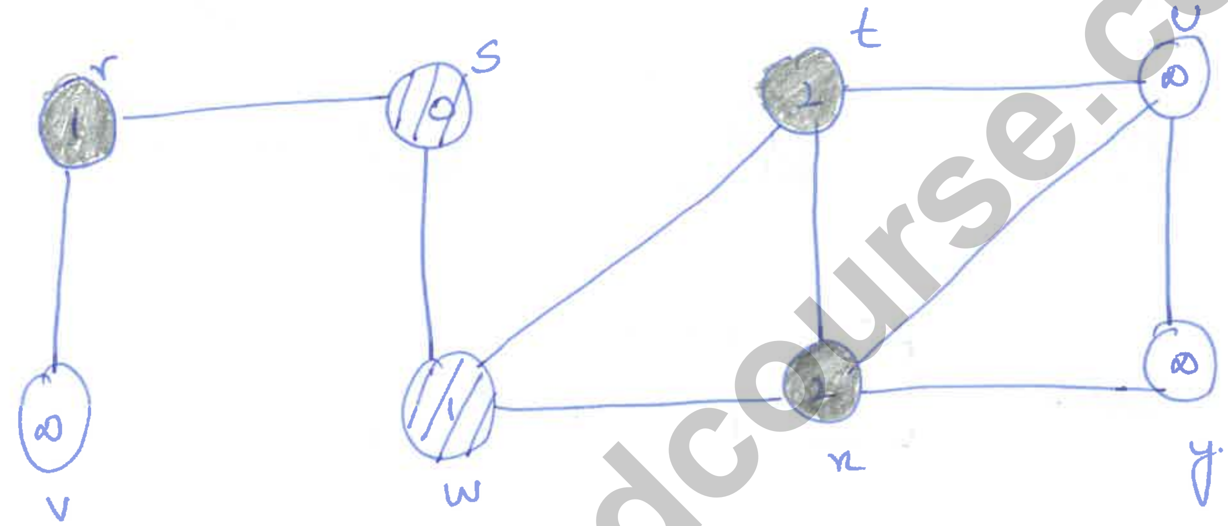
Their distance is updated as 0 + 1 = 1



Q

| | |
|---|---|
| | r |

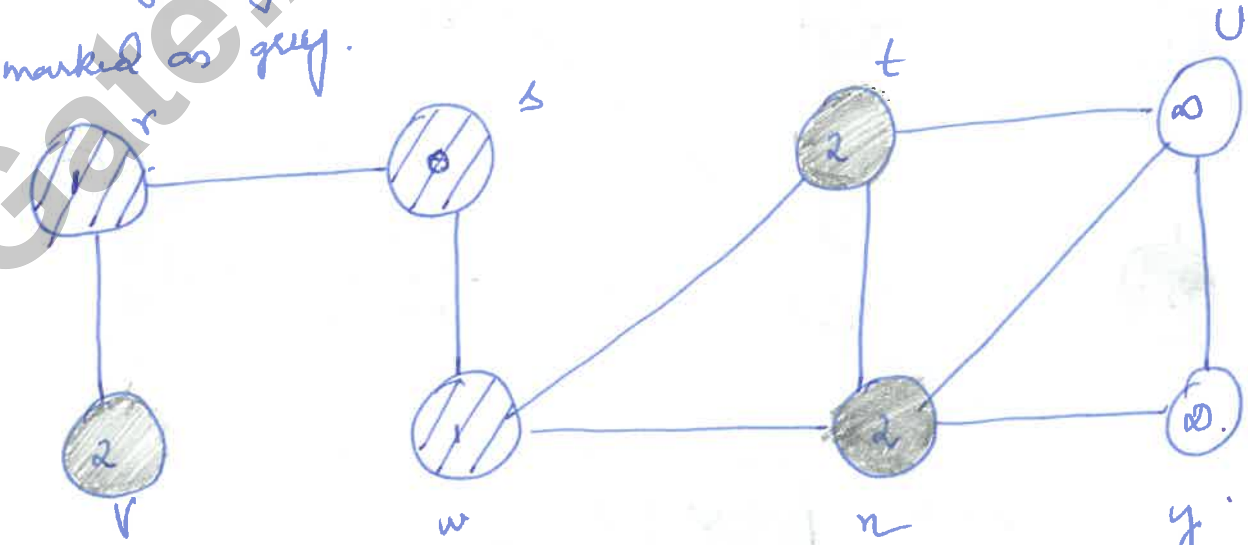
Now w is dequeued and its is marked as black and its neighbouring vertices i.e. {t, x} are added to the queue and their distance is updated as 1+1 = 2 and they are marked as grey.




Q

| | | |
|---|---|---|
| r | t | x |

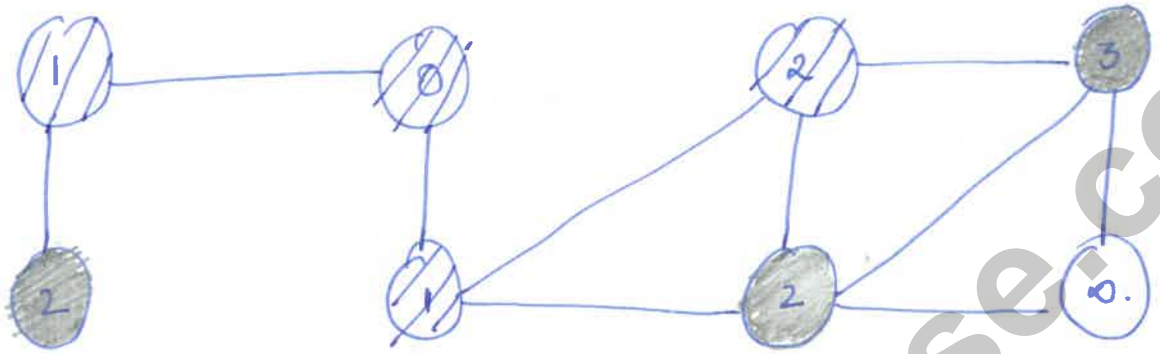
Now r is dequeued and marked black. Its neighbouring vertices i.e. {v} is updated as 1+1 = 2 and marked as grey.




Queue

| | | |
|---|---|---|
| t | x | v |

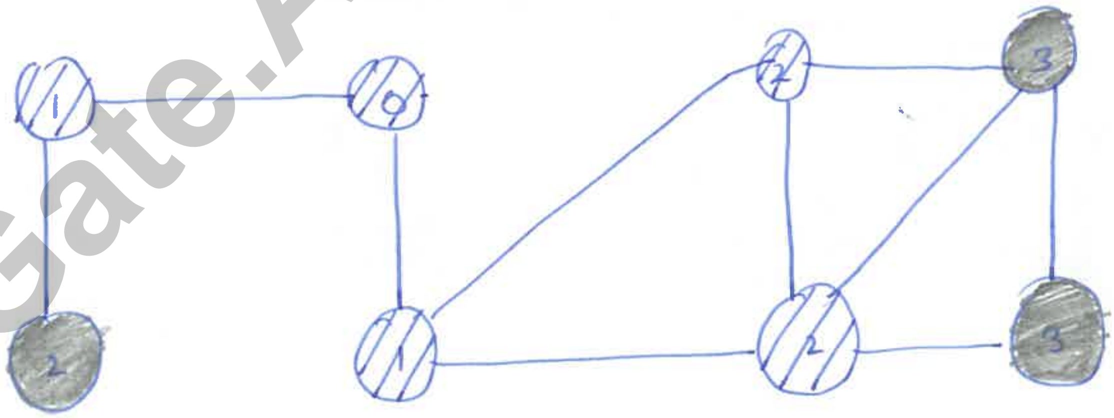
- Now t is dequeued and marked as black.
- Its neighbours {u} is marked grey and added to the Queue and the distance is labelled as 2+1 = 3.




Queue

| | | |
|---|---|---|
| x | v | u |

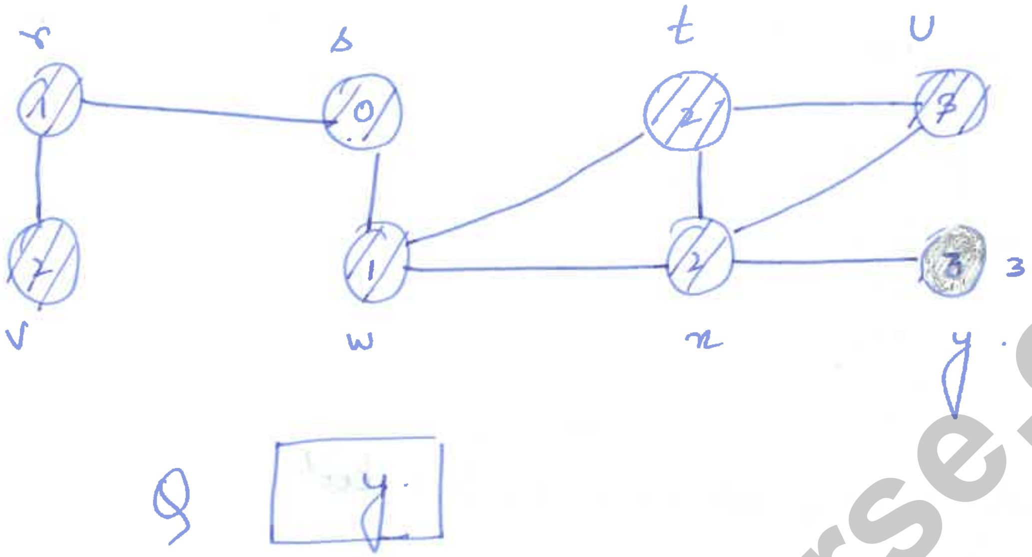
- Now x is dequeued and marked as black.
- Its neighbours {y} is marked as grey and it is added to the queue and its distance is updated as 2+1 = 3.



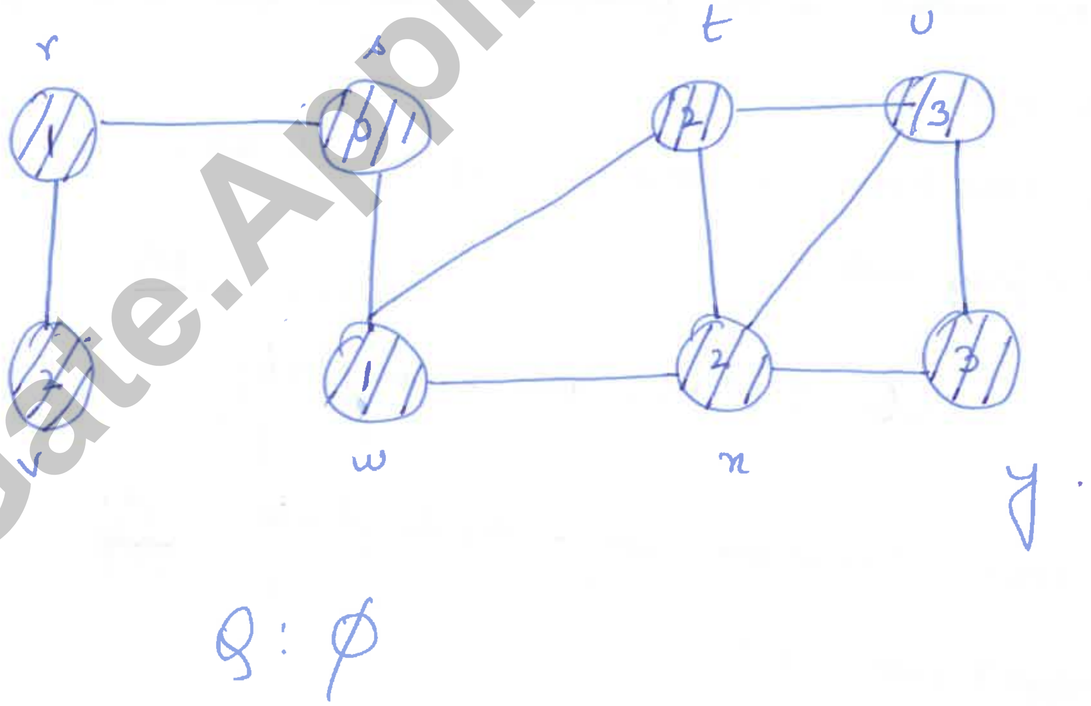

Queue

| | |
|---|---|
| v | u |

- Now u is removed from the Queue and it is updated as black
- It has no white neighbours so none are updated.




- Now y is dequeued and is updated as black.
- It has no neighbours as white so none are updated
- Now the queue is empty and the algorithm is completed.



# 60.7 BFS Color Coding Intuition

- Any vertex during the BFS is either white, gray or black.
- White is represented as 
- Gray as 
- Black as 

- This color coding is only used in CLRS text book.
- A white vertex means that this particular vertex is not yet visited.
- A grey vertex is a one which is present in the Queue, this vertex has been visited but all its children/neighbouring vertices are not visited.
- A black vertex is a vertex which is visited and all of its children have been visited and this particular vertex is also removed/dequeued from the queue.
- In some other books we may use a separate set called visited for black and grey nodes.

# 60.8 BFS: Code & Complexity

→ Let us discuss the code and analyze the complexity.

→ Code snippet from CLRS.

BFS( G, )

```
1.  for each u ∈ G.V - {s}
2.        u.color = WHITE
3.        u.d = ∞
4.        u.π = NIL
5.  s.color = GRAY
6.  s.d = 0
7.  Q = ∅
9.  ENQUEUE(Q, s)
10. While(Q ≠ ∅)
11.       u = DEQUEUE(Q)
12.       foreach v ∈ G.Adj(u)
13.            if v.color == WHITE
14.                 v.color = GRAY
15.                 v.d = u.d + 1
16.                 v.π = u
17.                 ENQUEUE(v)
18.       u.color = BLACK.
```



→ lines 1 to 9 are initalizations

→ All the distances for all the vertices except for the source are intialized to ∞.

→ for the source it is initalized to 0.

→ All vertices are marked as white and the source is made gray.

→ Sources of all the vertices are marked as NIL (Sources are nothing but predessor).

→ From lines 10 - 18 we have the while loop. where we are dequeuing a vertex from the queue and exploring all its white neighbours as gray and their distance is updated as the source distance +1. The source or predessor of the vertex is updated. 

- The source / predessor information helps us trace back the shortest path or the BFS path to the source vertex.

- Once all the white or unexplored neighbours are explored the dequeued node is marked as black (line 18).

- Time Complexity.

- Lines 1-4 are executed n times ($n = |V|$) - $O(n)$
- Lines 5 - 9 are executed only once - constant time - $O(1)$.
- In lines 10-18 :-

  We should note that every node will get enqued and dequeued only once (n times)

  - Time complexity to enque and deque = $O(1)$.

  - In the for loop of lines 12 - 17, we are processing it based on the adjacency list, no of elements in the adjacency list is exactly equal to the number of edges - $O(m)$ ($m = |E|$).

    - Within the loop we are doing constant time work $O(1)$ for lines 13 to 17.

- Now total time complexity

$$\underbrace{O(n)}_{\text{Initialization}} + \underbrace{O(n)}_{\text{No of times the while loop executes deque \& enqueue operations}} + \underbrace{O(m)}_{\text{No of times for loop execution}} = \underline{O(n+m)}$$

- When the graph is a dense graph then $m = O(n^2)$, then the time complexity becomes $O(n+n^2) = O(n^2)$

- In case of a sparse graph when $m << n^2$ then time complexity is $O(n+m)$.

Space Complexity

→ If we consider the space required for the Adj list then it will take up $O(n+m)$.

- Space required to store the color coding info and predecessor info and for the Queue = For each vertex we require 1 location for color info and 1 location for predecessor information and maximum possible length of the queue is n.

$= O(n) + O(n) + O(n)$

$= O(n)$.

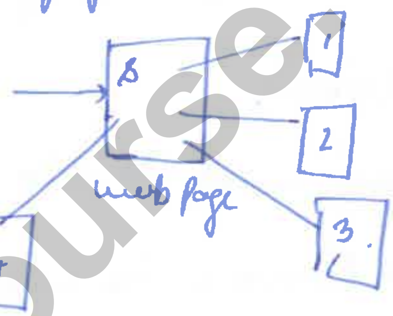
If we ignore the space for Adj list (if it is taken as an i/p) then space complexity $= O(n)$.

- If we include the space for Adjacency list then it is $O(n+m)$.

# 60.9. BFS Applications

1. BFS can be used to find the shortest path from a source to all the other vertices of the graph in an unweighted graph (graph with equal weights).

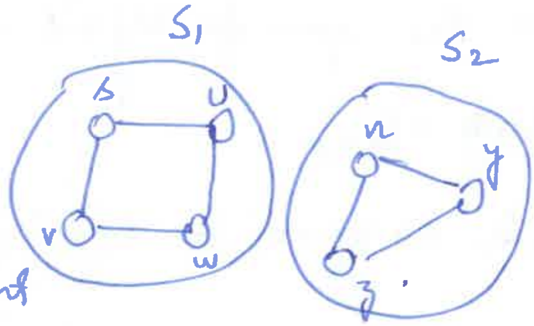
2. **Web Crawlers :-** Web Crawlers use BFS to crawl the web pages of the different websites by using the web graph.




3. **Social Network :-**

- In social networks we can use the BFS to find how far a person is from another (s to v) to find the distance and recommend friends.


4. **Connected Components :-**




If we perform BFS on the graph with s as the source all the vertices of the connected component belonging to s are traversed i.e component $S_1$.

- But all are not traversed if a check is performed on the total no of vertices of the graph and the vertices traversed if it is less then we can come to know that we have more than one component of the graph.

- We need to traverse again using any of the remaining vertices {x, y, z} as the source to discover the other component of the graph. i.e {x, y, z}.

→ We can know how many connected components we have and what are the vertices in each component by revisiting the graph or re applying BFS until all the vertices are traversed/visited.




---

## 6.0.10 Depth First Search.

## Intitution & Code.

- Another way to traverse a graph other than BFS.

```
DFS(G)
1.  for each vertex u ∈ G.V
2.        u.color = WHITE
3.        u.π = NIL
4.  time = 0
5.  for each vertex u ∈ G.V
6.        if u.color == WHITE
7.              DFS-VISIT(G,u).
```

```
DFS-VISIT(G,u)
1.  time = time + 1
2.  u.d = time
3.  u.color = GRAY
4.  for each v ∈ G.Adj[u]
5.        if v.color == WHITE
6.              v.π = u
7.              DFS-VISIT(G,v)
8.  u.color = BLACK
9.  time = time + 1
10. u.f = time
```

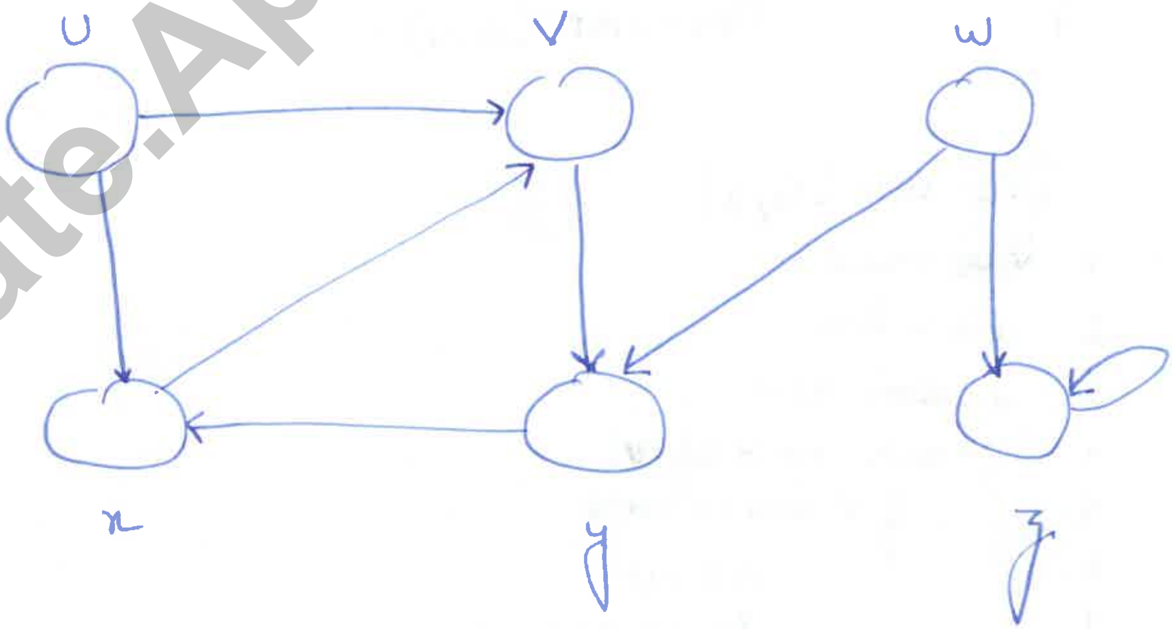
→ Because we have recursive function calls for DFS-VISIT(), recursion makes use of stack internally, this implementation can also be re written without recursion and by using a stack explicitly.

→ Lines 1-4 are initialization where each vertex is coloured white and the source of each vertex is marked as Nil.

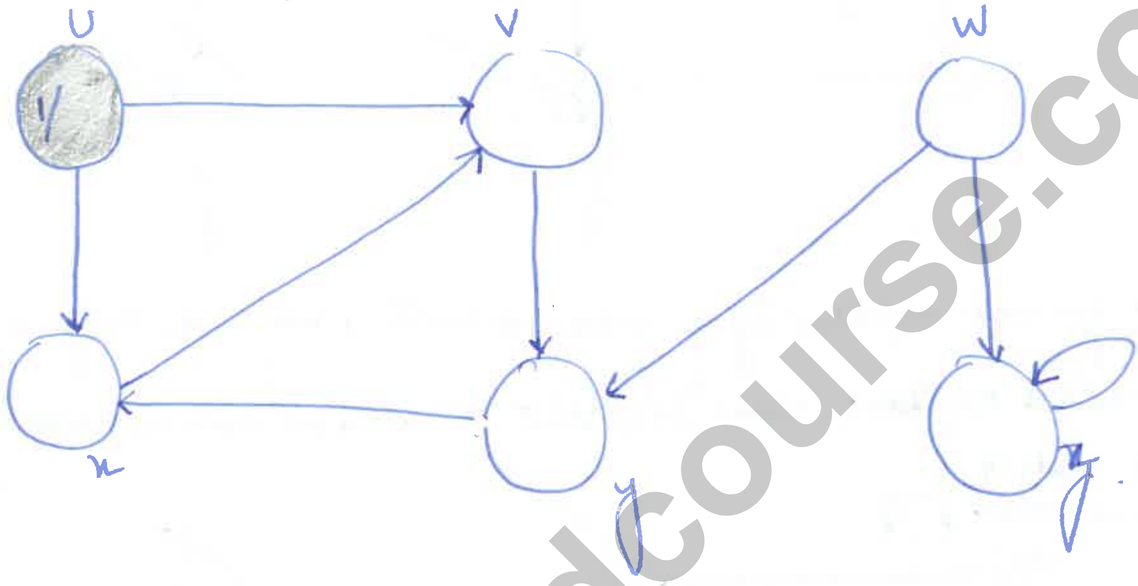
→ From lines 5-7 If the color of a vertex is white then the functions DFS-visit is called for that vertex.

→ The DFS-VISIT function records the time when a vertex is first explored and it marks the vertex as GRAY, Once it is marked as GRAY it is then used to explore all its white neighbours for each of its neighbours the DFS-VISIT is called recursively, once the DFS-visit is called on all the neighbouring vertices then this current vertex is marked as Black and this time is also recorded as the final time of the vertex.

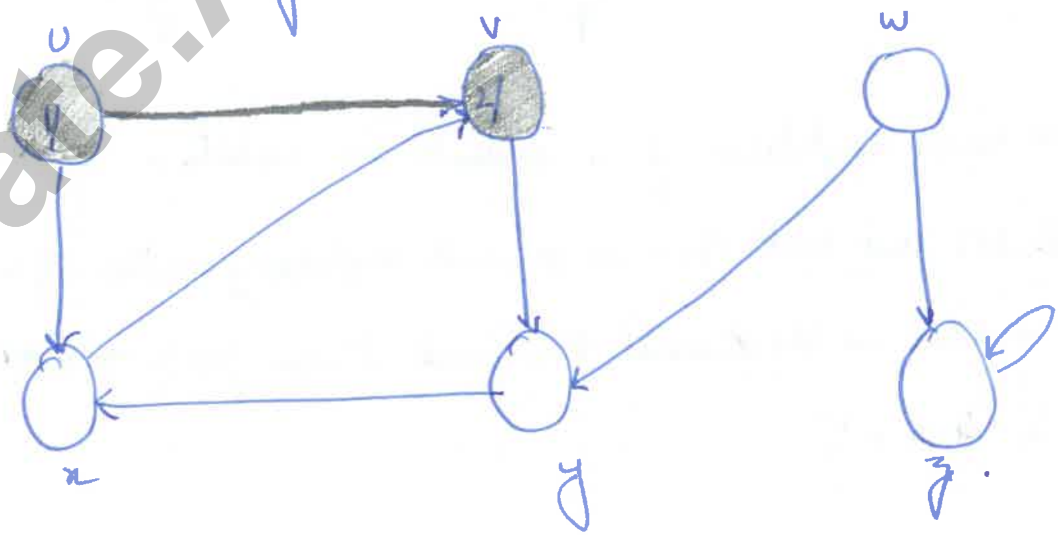
→ Let us consider the example of the following digraph for DFS.



- Initially lets assume DFS Visit is called on node u
- ~~It is ma~~
- The time variable is initialized = 0
- time = time +1 = 0+1 = 1
- U.d = 1
- It is coloured Gray

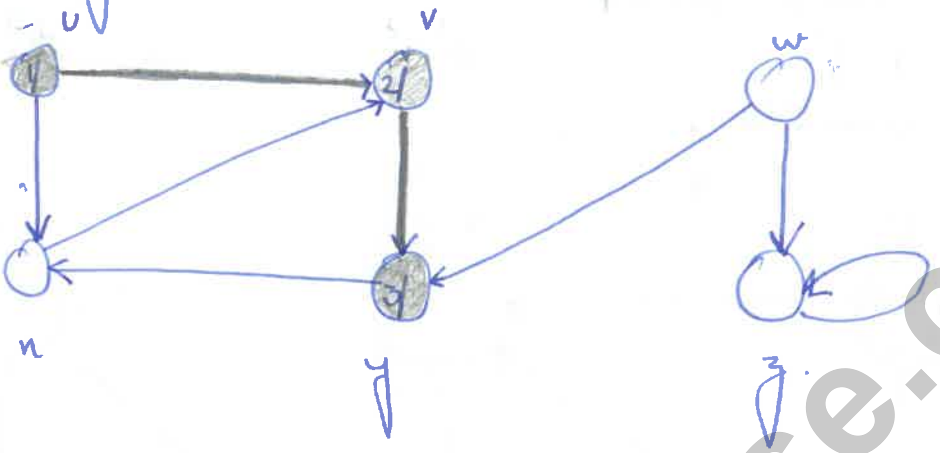



- Now for each neighbour of U DFS-VISIT is called., we can call on x or v
  let us call on v.
- time = 1+1 = 2.
- v.d = 2
- V is coloured Gray.



→ Now for each neighbour of v which is white we have y as the only such neighbour, DFS-VISIT is called on y recursively.

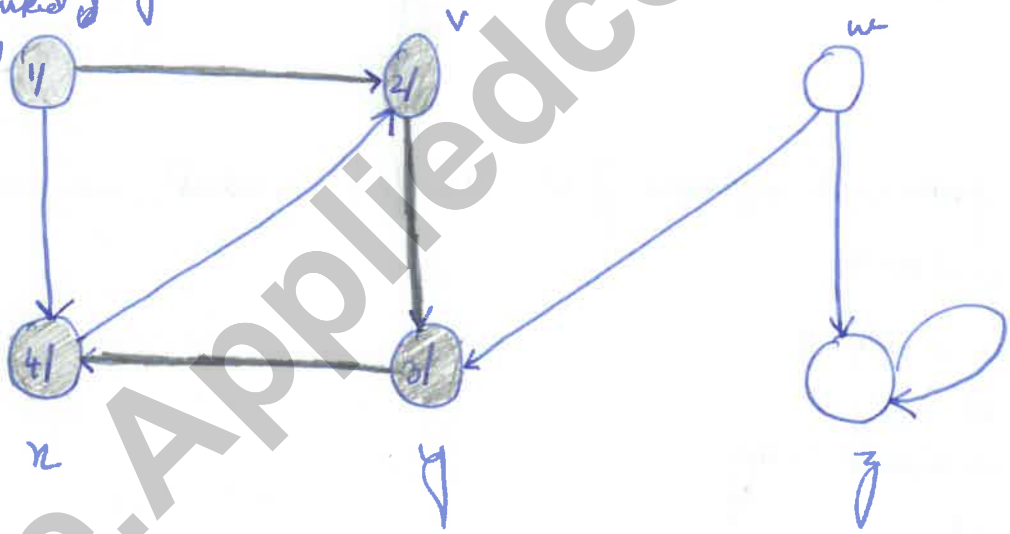
- time = 2+1 = 3
- y is marked Gray



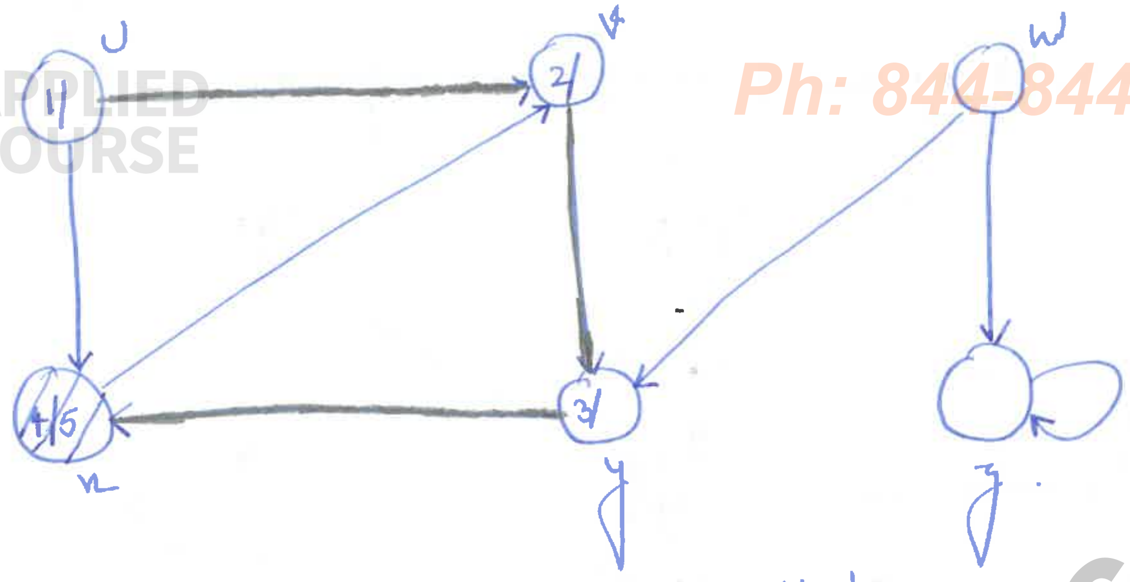

→ Now for each neighbour of y which is white, we have only x as one such a neighbour, Now DFS-VISIT is called on node x recursively

time = 3+1 = 4

x is marked grey




→ Now for each neighbour of x which is white, we need to call DFS-VISIT but there is no such neighbouring vertex of x, now x is marked as black and the final time for x is marked to as time = 4+1 = 5

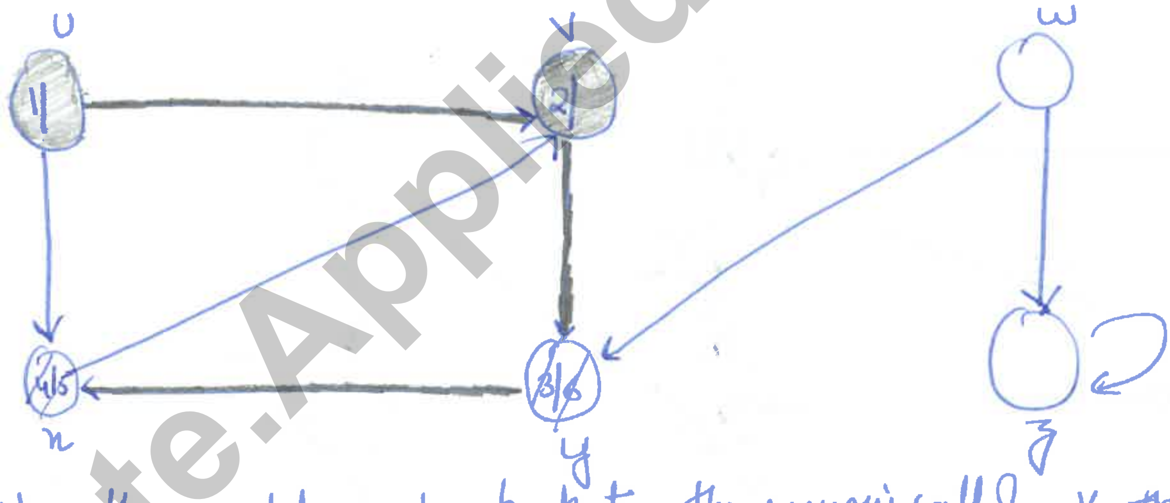



→ Now the control reaches to the recursive call of y.

→ If any neighbouring vertices of y are white then DFS Visit is called, but no such vertex exists.

→ Now y is marked as black and the time is updated time = 5+1
= 6.

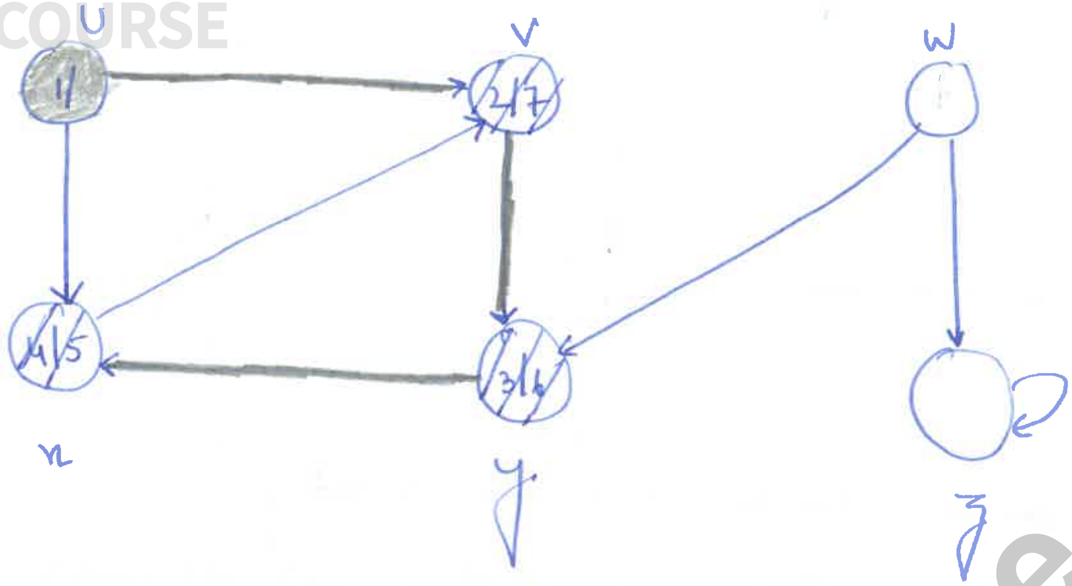
the final time for y = 6.




- Now the control reaches back to the recursive call of v, ~~the~~ it is checked if there are any white neighbours of v, we are not able to find any such.

→ Now v is marked as black and time is incremented = 6+1
= 7.

and the final time of v is marked as 7.




→ Now the control returns to the call DFS-Visit of vertex U

→ Any neighbouring vertices which are white are checked for, but no such vertex exists, now U is marked as black and time is incremented time = 7+1 = 8.

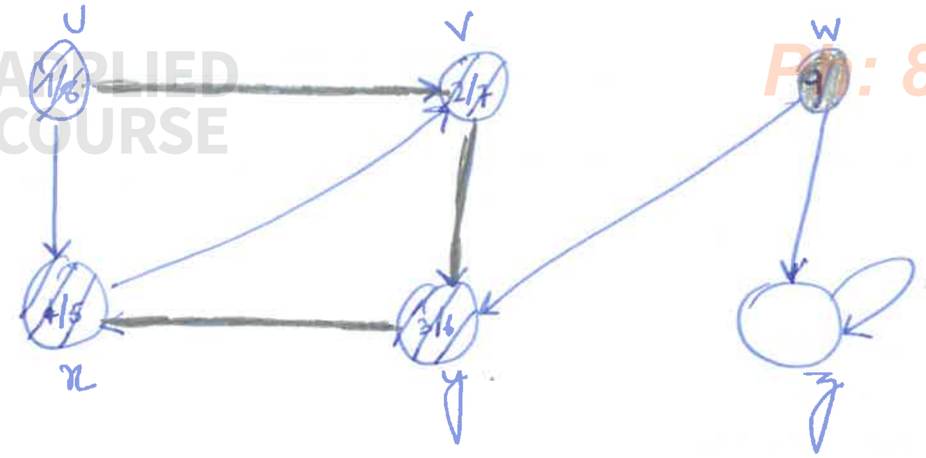
– Final time for U is recorded U.f = 8.


– Now the control returns to the DFS function and other vertices which are white are checked for. W is picked and DFS-Visit is called on w, now

– time is incremented = time = 8+1 = 9

– w is marked as grey.

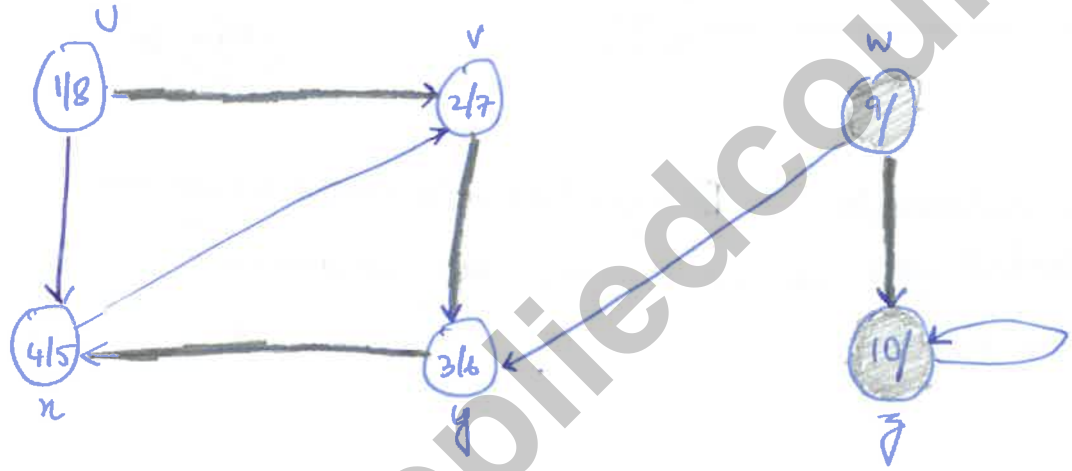
– time for w is assigned = 9




→ Now nodes neighbouring to w are looked for which are white, z is only such vertex, DFS-VISIT is called recursively on z

→ z is marked grey and time is incremented = time = 9+1 = 10.

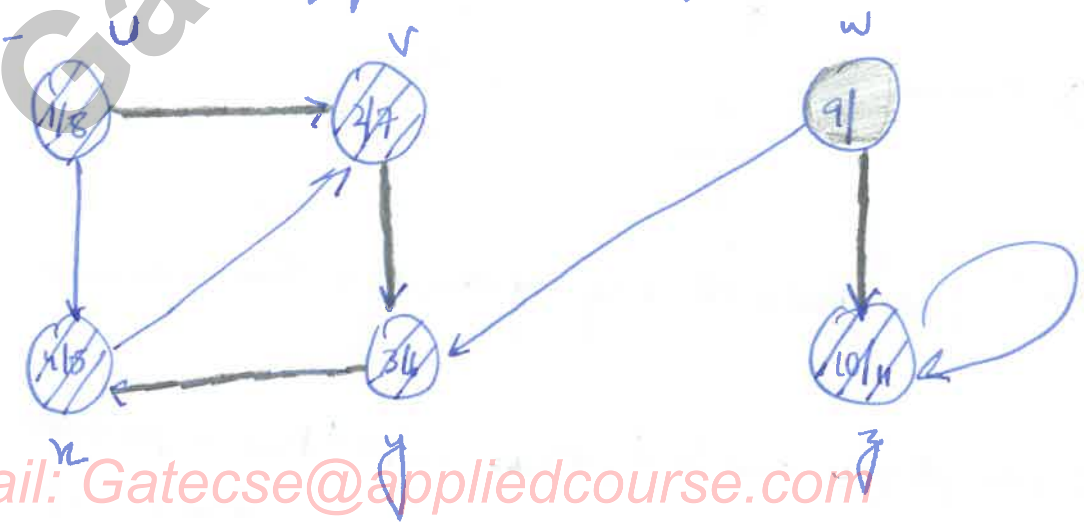
→ Initial time for z is marked as 10



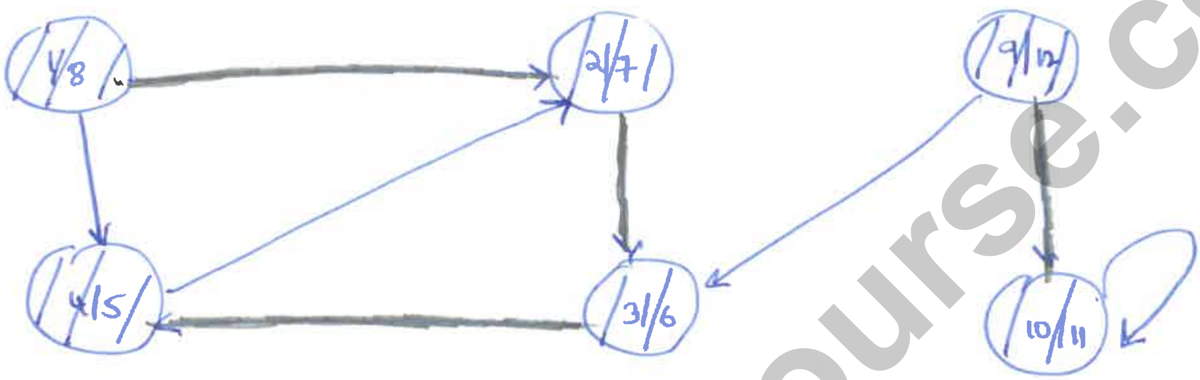

→ Now nodes neighbouring to z which are white are looked for, no such node exists so now z is marked black and the time is incremented

time = 10+1 = 11

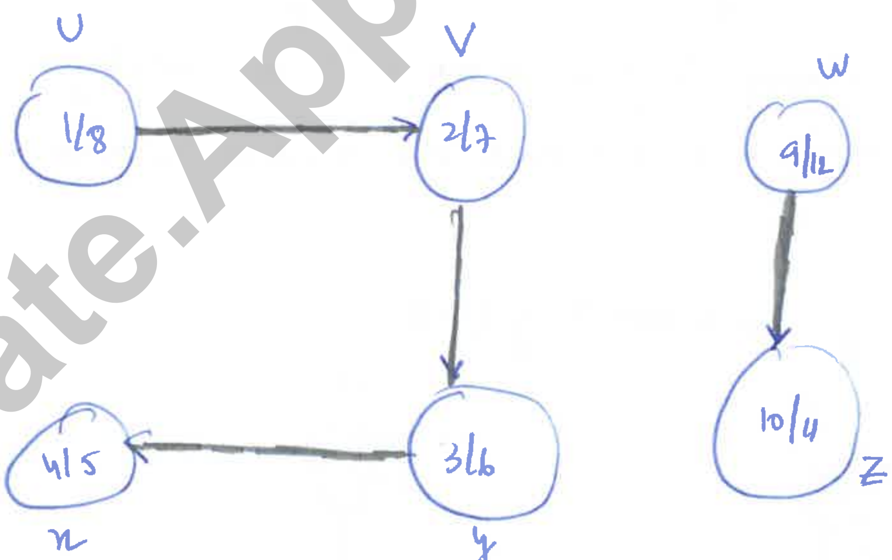
- The finish time of z is marked as 11 z.f = 11



- Now the control returns to the recursive call of DFS Visit of node w
- Nodes neighbouring to the node w are looked for which are white, no such node is present.
- w is marked as black.
- time is incremented time = 11+1 = 12.
- final time of w is marked w.f = 12.




- Now the control returns to the DFS function and nodes which are white are looked for, no such nodes are remaining.
- The DFS comes to an end.




- The above is graph constructed only by reminding the predecessor edges.
- Both of them we get two trees, these are DFS trees or predessor trees.

- An edge from predecessor to successor in the DFS traversal is known as predecessor edge. (U,V), (V,y), (y,x), (W,z) are predecessor edges.

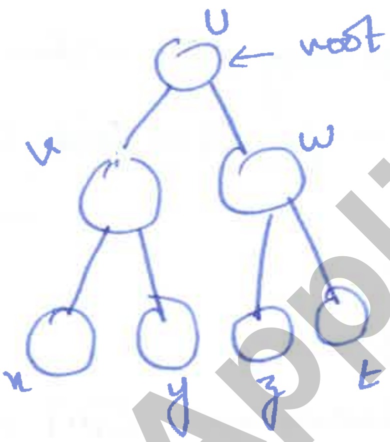
- Tree edges are also those edges which are part of the DFS tree of the graph.

- An edge from one tree to another tree in DFS tree's is known as a **cross edge**. (w, y) is a cross edge.

- **Forward Edge** is an edge from a vertex which is visited earlier to a vertex which is visited later. example (U, x)

- **Back Edge** is an edge from a vertex which is visited later to a vertex which is visited earlier in DFS. for example (x,V)

- **Comparison b/w BFS and DFS.**




In BFS: At vertices at distance 1 from root are explored first, then those vertices that are at distance 2 and so on, we traverse breadth first.

U, V, W, x, y, z, t.

- In DFS the traversal is depth first, we try to go as deep as possible first and then return

U
V
x
y
w
z
t

Depth first.

→ BFS makes use of a queue, DFS if implemented recursively makes use of the call stack if implemented iteratively it makes use of an explicit stack.

## 60.11 DFS Analysis



- Time complexity for DFS.

→ Lets assume $n = |V|$
$m = |E|$

→ Lines 1-3 are executed for each vertex $O(n)$ time work is done.
→ Lines 3-7 are executed for each vertex $O(n)$ even though the DFS-VISIT Function is called recursively, the DFS Visit function is called exactly once for each vertex as it is called only for white vertices and once it is called then the vertex is coloured gray.

→ In the DFS-VISIT function which is called for every vertex in lines 4 to 7. we visit their neighbouring vertices. This will run for $O(m)$ times it will run no of edges times.

- Total time complexity $O(n)+O(m) = \underline{O(n+m)}$.
- Space complexity :- Similar to BFS only additional space $O(n)$ for colour coding and time to store.

## 60.12 DFS Edge Types.

Types of Edges in DFS :- On Running DFS we may get one or multiple DFS trees. The collection of DFS trees is also known as DFS forest.

① Tree Edges :- An edge that is part of the DFS forest is the DFS tree.

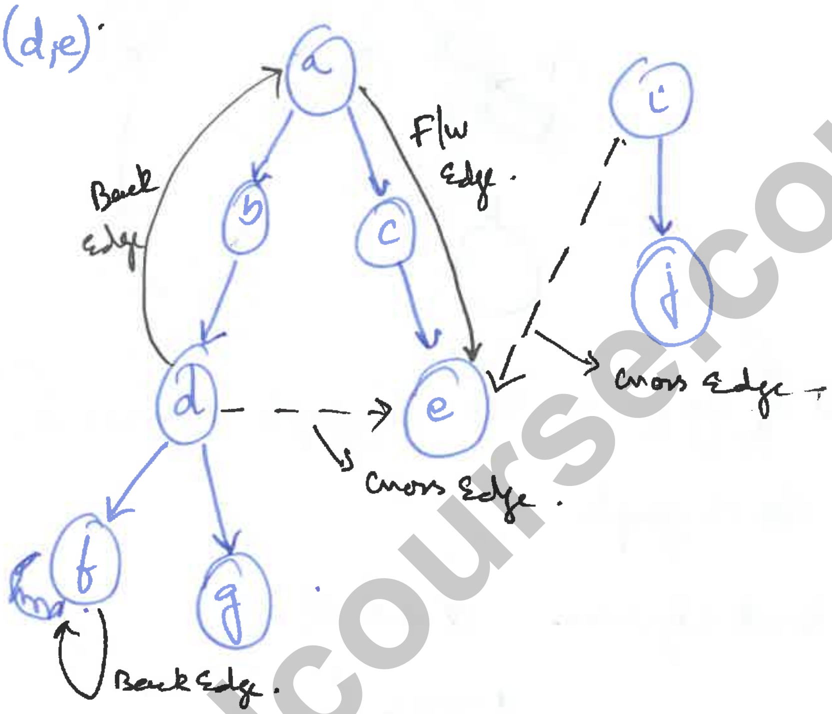
② Back Edge :- If Vertex U is a predecessor of V then (U, V) is a Back Edge, or self loops.

③ Forward Edge :- Edge (U,V) not part of the forest, U is an ancestor and V is a descendent.

(4) Cross Edges: - (a) Edge b/w different trees. example (i,e)

(b) (u,v) ∉ DFS-forest no relation b/w ancestor or descendant.

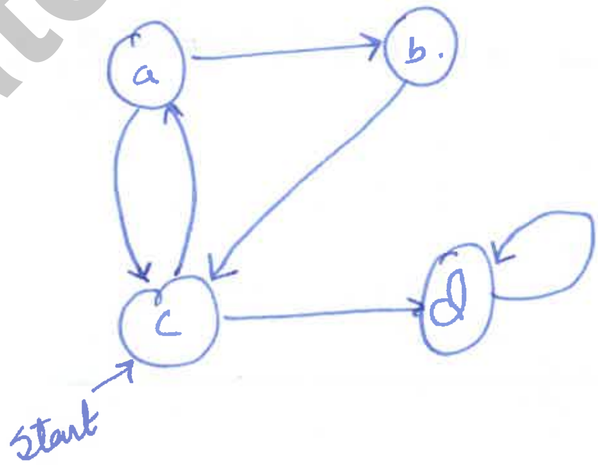
example (d,e).



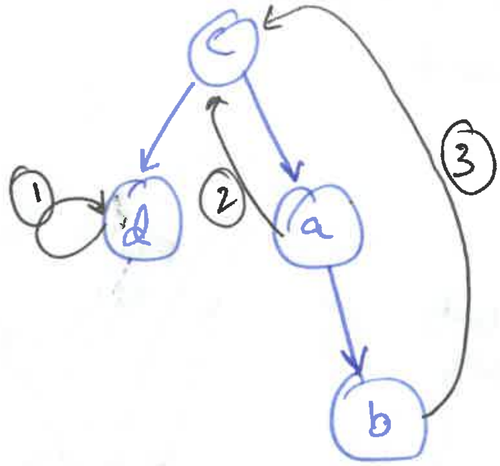

## 60:13 Applications of DFS:- Detect Cycles in a di-graph.

→ By using DFS we can ~~use it~~ detect cycles in a di graph.

→ Back Edges are useful in detecting the cycle in the graph.



The DFS tree/forest is as follows. 




# of cycles in the di-graph is equal to the no of cycles in the di graph.

The back edges are
$d \to d$ (1)
$d \to c$ (2)
$b \to c$ (3).

→ How do we detect a back edge during the DFS function.

during DFS if we encounter an edge (U,V) such that V is gray then (U,V) is a back edge. and there is a cycle corresponding to it.

– Time complexity to determine a cycle = Time complexity of DFS = $O(V+E)$
$= O(V+E)$

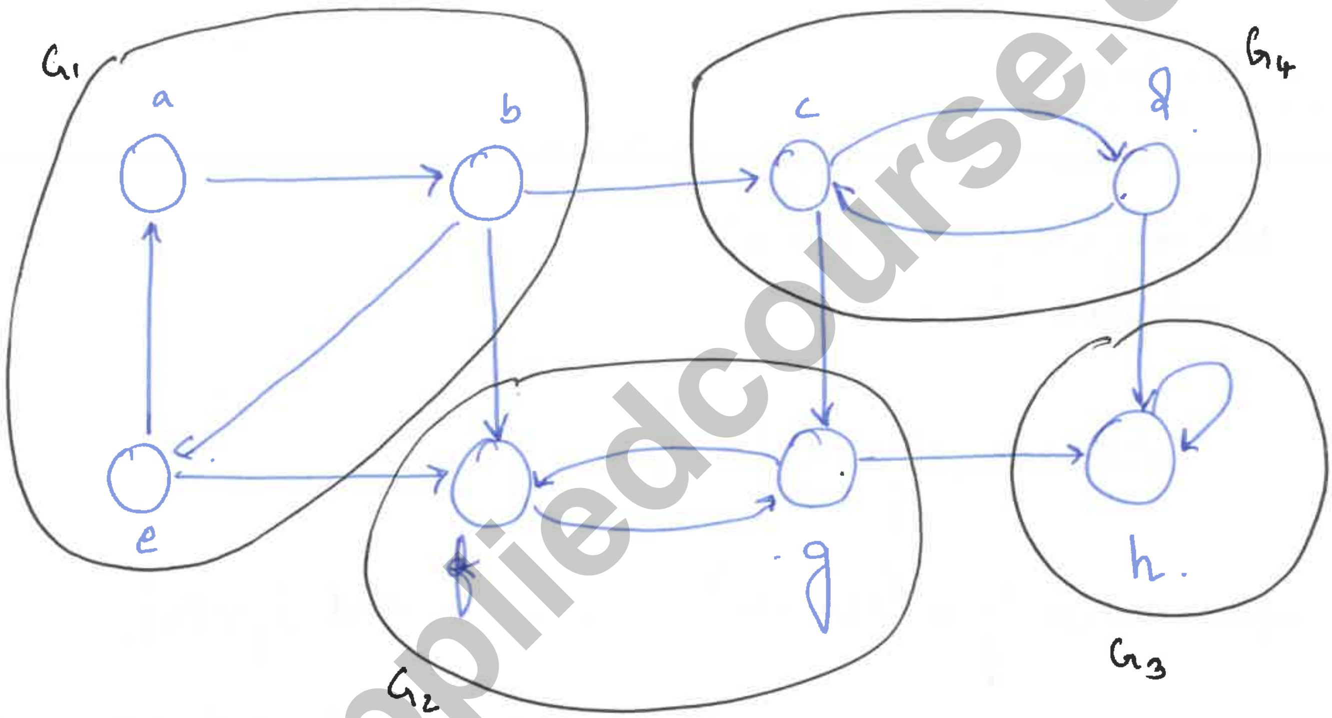
# 60.14 Application: Strongly Connected Components



Definition of Strongly Connected Component :- SCC of a graph G is that subgraph G' such that

1. G' has path $u \rightsquigarrow v$ $\forall$ vertices $u, v \in G'$

2. G' is maximal set of vertices that satisfy property (1) above.






- The above graph has 4 strongly connected components.

- SCC have multiple applications for example a web graphs on a website but different web pages have been represented as nodes and different hyper links on pages are represented as edges, SCC's would represent related pages for example for different pages on wikipedia.

- The algorithm using DFS is known as Kosaraju's Algorithm

(1) Compute DFS(G) and finishing times (v.f) for each $v \in G$ is noted.

(2) $G^T$ (Transpose Graph) G: (U)→(V) $G^T$: (V)←(U) Edges are reversed.

If G is given as adj list $G^T$ can be computed in $O(n+m)$ time

(3) Call DFS($G^T$) with specific ordering of vertices in decreasing order of v.f. recursively.

(4) Output of vertices of each tree in the DFS forest, each tree is a SCC.

Time Complexity

DFS is called twice = $O(n+m) + O(n+m)$

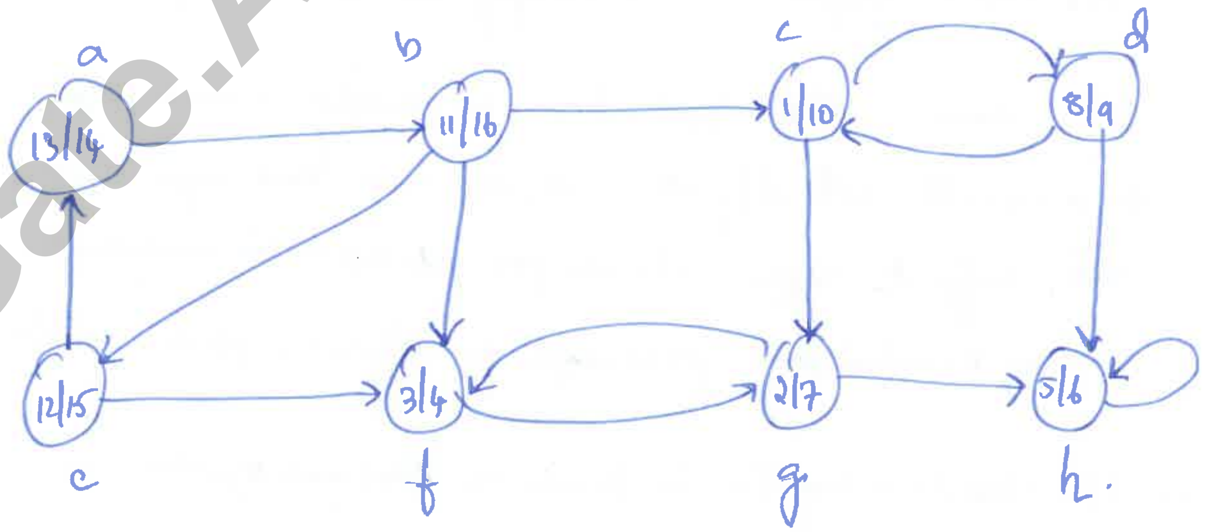
$\therefore$ Taking note of v.v = $O(n+m)$

and finding max.

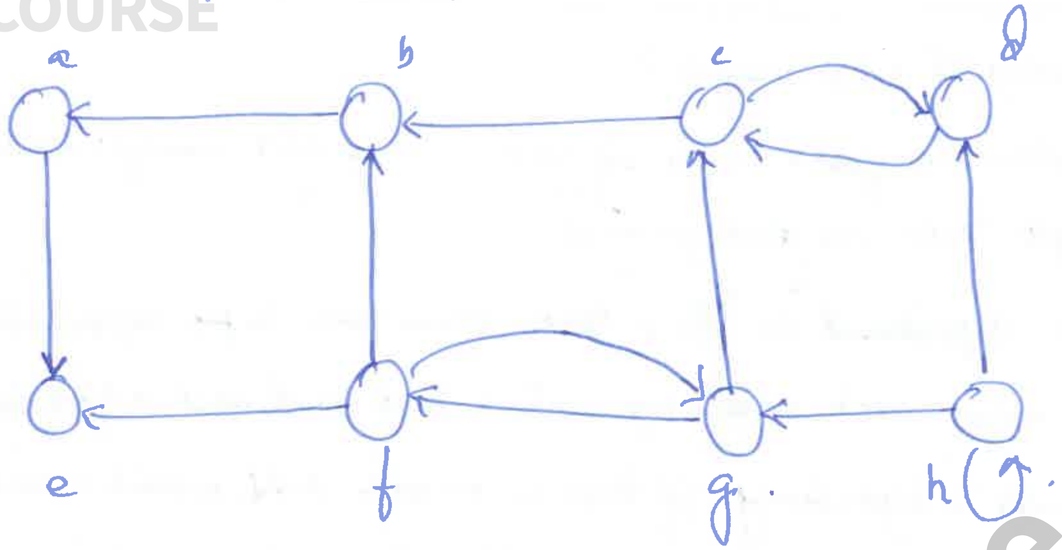
Total time complexity = $O(n+m)$

Space Complexity = $O(n+m)$ (same as that of DFS).

Example of above graph if we do DFS we will get the following d/f values.



The reversed graph is as follows.




Now we need to perform DFS of the above $G^T$ graph on starting from the node with more finish time value.

Starting from b.

b to a to e.

we have $\{b, a, e\}$ as one component. $C_1$

→ Now next max vertex c

Starting from c we can reach d and no other vertex

$\{c, d\}$ is another component $C_2$.

→ Now next max vertex = g.

Starting from g we can reach f and no other vertex.

$\{g, f\}$ are another component $C_3$.

→ Next max vertex is h.

- From h we cannot explore any other vertex.

$\{h\}$ is another component $C_4$.

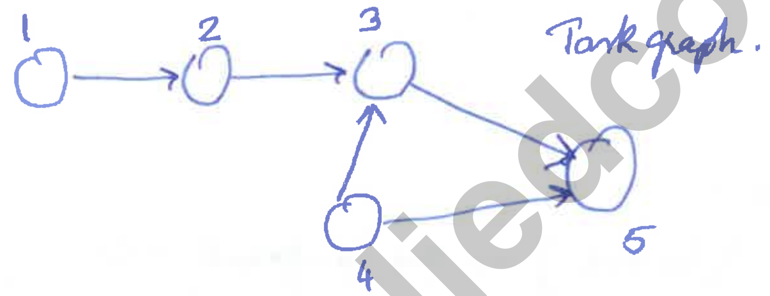
# 60.15 Application of DFS : Topological Sort



- How is topological sort useful?

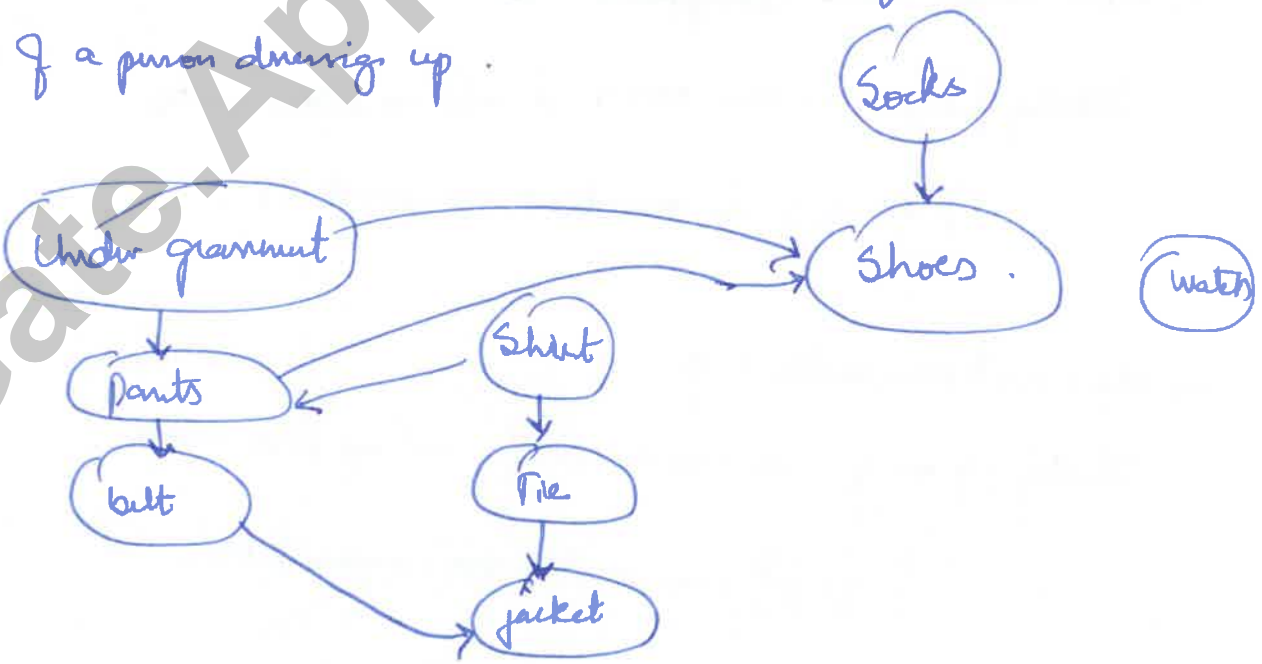
It has many applications for example if you are a project manager and there are multiple tasks in your project

Some tasks are dependent on other tasks these can be ~~pre~~ represented by a graph, where each node represents a task and each edge (directed edge) represents a dependency if there is an edge U→V which means that the task V has a dependency on task U and task ~~you~~ U should be completed before task V is ~~completed~~ started




→ Topological sort provides us with the logical ordering of the tasks

- Example of a person dressing up.




Ordering: Socks, Under Garments, pants, shoes, watch, shirt, belt, tie, jacket.

→ There could be more than one ordering possible in topological sort.



## Topological Sort

1. Call DFS(G) & Compute u.f for each vertex u.
2. As each vertex finishes, insert the vertex @ front of a linked list
3. return the linked list.

## Time Complexity

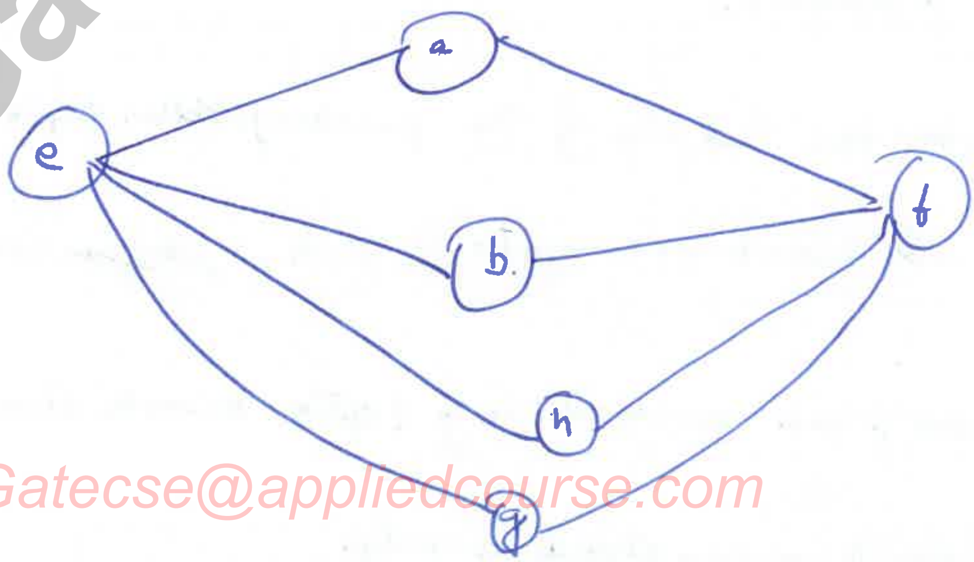
1. For step 1 $O(n+m)$
2. Step 2 Insert into a linked list is $O(1)$ time into is called $O(n)$ times ∴ total time complexity $O(n)$
3. Step 3 :- $O(1)$ time

∴ Total time complexity $= O(n+m)$.

---

## 60.16 Solved Problem Gate 2003

Q). Consider the following graph



Ph: 844-844-0102

Among the following sequences

I) a b e g h f.

II) a b f e h g.

III) a b f h g e

IV) a f g h b e.

Which are depth first traversals of the above graph?

1. If we start DFS from a as all options have a as the starting vertex

2. a we can choose e, b or f. from these let us choose b (as many options have b)

3. a-b from b we can we can chose e, h, or f, lets choose e.

4. a-b-e now from e we have only g, lets choose g.

5. a b e g from g we have h, f. lets chose h (as it is present in the 1st option).

6. a b e g h from h. we can visit only f

7. a b e g h f - I matches.

Lets continue from the 2nd step of the previous DFS exploration.

1. a-b is explored and from b we can go to f, e, h, lets choose f

2. a-b-f from f we can choose h, g (option II can be eliminated) lets choose h

3. a-b-f-h from h we can choose g only.


Mail: Gatecse@appliedcourse.com

4 a-b-f-h-g from g we can choose e only.

5. a b f h g e III option is correct.

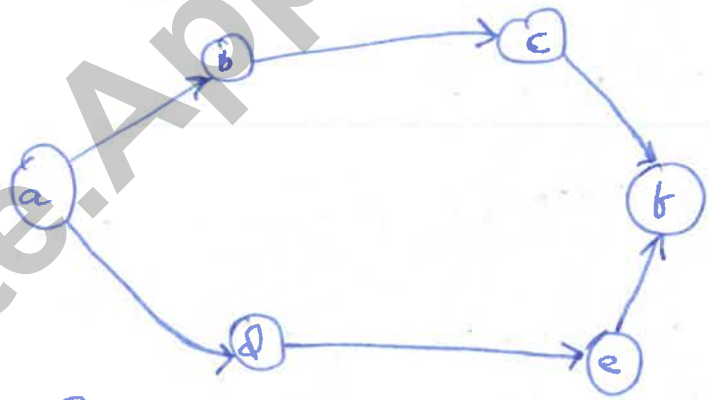
---

- We are left with option IV lets verify
  - b Starting from a we can choose f.
  - a-f from 'f' we can choose b, h, g. lets choose g
  - a-f-g from 'g' we can go to h, e lets choose h
  - a-f-g-h from 'h' we can go to b only.
  - a-f-g-h-b from 'b' we can go to e only.
  - a-f-g-h-b-e now this matches with option IV

So I, III and IV are correct option D ✓

---

60.173 Solved Problem Gate 2016




The number of different topological orderings of the vertices of the graph is ___ ?

a has to be the first and f has to be the last.

.a _ _ _ _ f. b has to come before c. and d has to come before e. } These 2 conditions should be preserved.

Ph: 844-844-0102

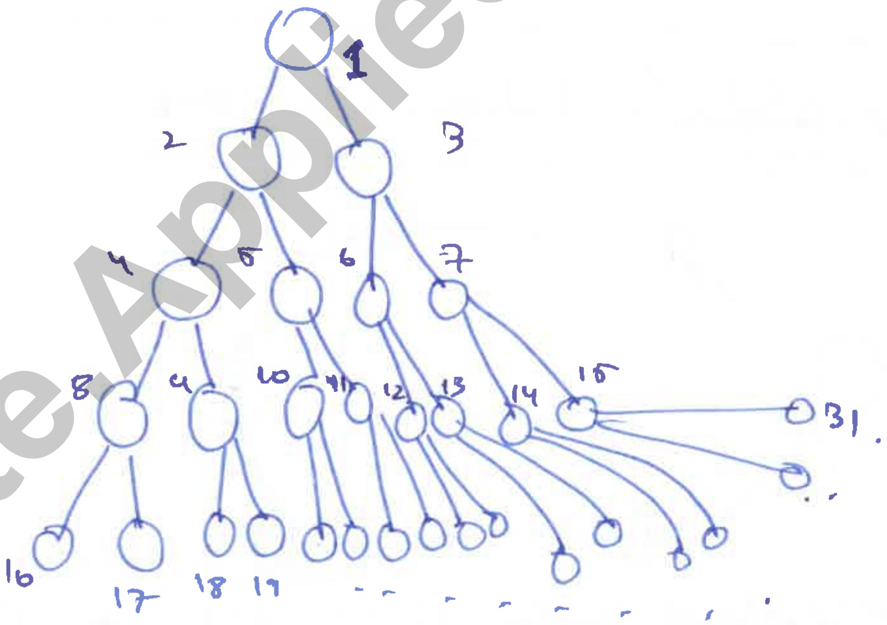
bc de
bd ce
bd ec

debc
db ec
db ce.

only these 6 possibilities will respect the conditions.

## 60. SOLVED PROBLEMS GATE 2016-1

Breadth first search is started on a binary tree beginning from the root vertex. There is a vertex t at a distance four from the root. If t is the nth vertex in the BFS traversal the maximum possible value of n is ____.




- As BFS will traverse in the order of the distance and to get the maximum we will consider a full tree.
- Now as the above diagram shows that one at distance 1 max possible value = 3.
- similarly for distance 4 maximum value = 31.


Mail: Gatecse@appliedcourse.com

## 60.19 Solved Problem Gate 2000

→ Let G be an undirected graph, Consider a depth-first traversal of G and let T be the resulting depth-first search tree. Let u be a vertex in G and let v be the first new (unvisited) vertex visited after visiting u in the traversal. Which of the following statements is always true?

(A) {u,v} must be an edge in G, and u is a descendent of v in T.

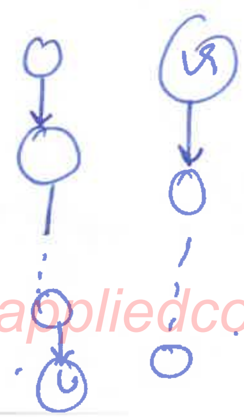
(B) {u,v} must be an edge in G, and v is a descendent of u in T

(C) If {u,v} is not an edge in G then u is a leaf in T.

(D) If {u,v} is not an edge in G then u and v must have the same parent in T.

– A option Need not true: If u and v belong to different trees, and u is explored first in the DFS exploration than v in this case u need not be descendent of v.

→ 'B option need not be true because if they are in different trees of DFS exploration but there need not be an edge from u to v.

→ C option T  This option seems correct. u is a leaf if uv is not an edge

→ D option need not be true, if (u,v) is not an edge then they could also belong to different trees as well.

---

## 60.20 Solved problem

Gate 2014 Set-1

- Let G be a graph with n vertices and m edges. What is the tightest upper bound on the running time on Depth First Search of G.? Assume that the graph is represented using Adjacency matrix.

(A) $O(n)$

(B) $O(m+n)$

(C) $O(n^2)$

(D) $O(m * n)$.

In the DFS-VISIT function we have a function call which for each neighbour of each vertex.

This loop executes $O(m)$ times which is no of edges, but if an adjacency matrix is being used to determine the neighbours of a particular vertex we need to traverse the complete row of the adj. matrix which will takeup $O(n)$ time for each vertex.

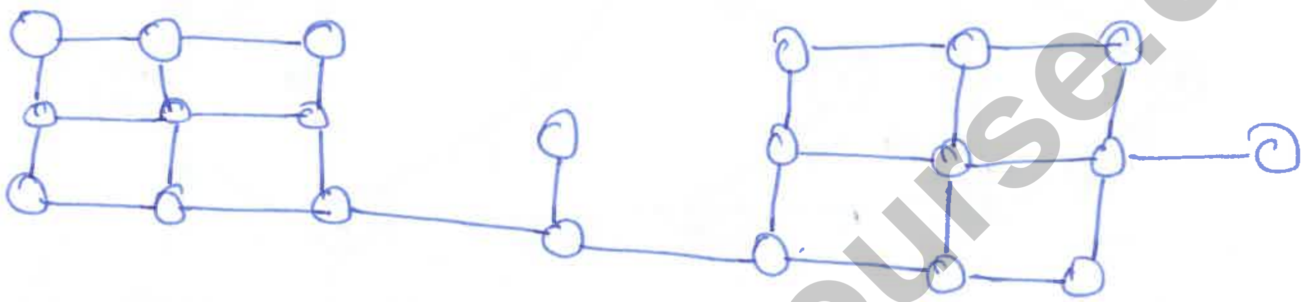
∴ Total time taken $O(n^2)$. option C.

Q0. 21. Solved Problem GATE 2014 Set 3



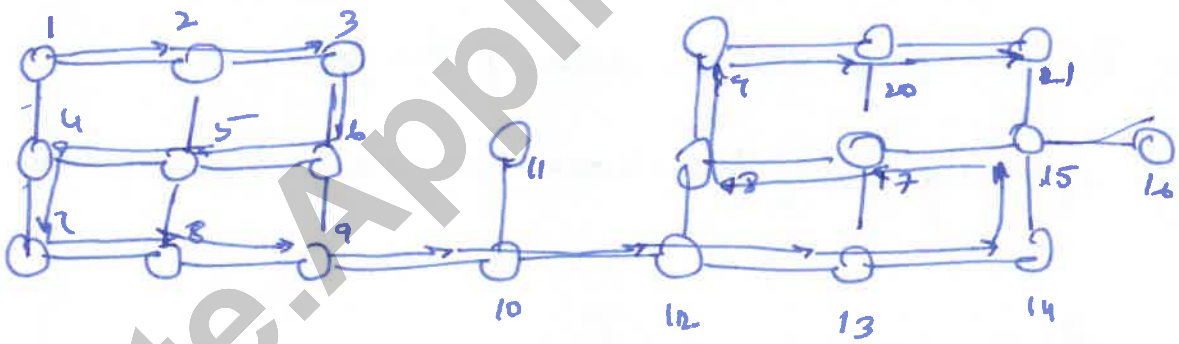
Suppose DFS is executed on the graph below starting from an unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth (including the initial call is) ____.

A) 17 B) 18 C) 19 D) 20.



We need to figure out the longest possible path.

Let us number the vertices




The longest possible path is

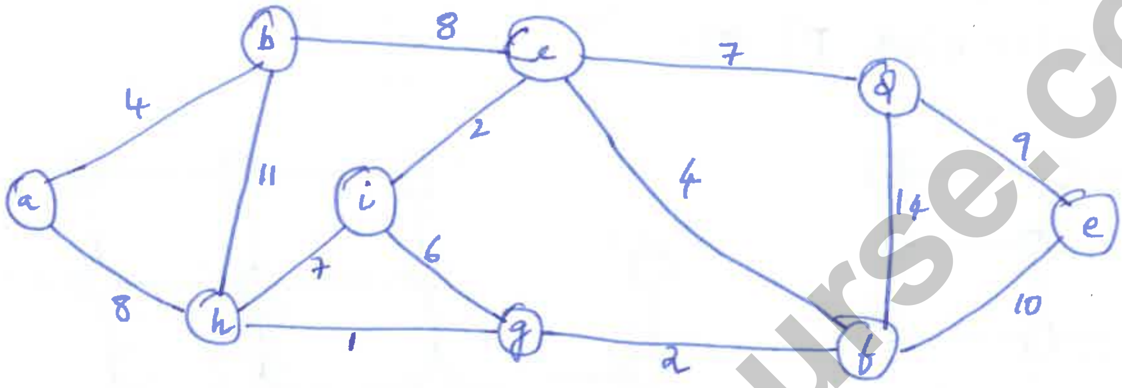
1 - 2 - 3 - 6 - 5 - 4 - 7 - 8 - 9 - 10 - 12 - 13 - 14 - 15 - 17 - 18 - 19 - 20 - 21

- The longest path corresponds to longest exploration path which have maximum elements in the call stack.
- The path length is 19 edges.
- Anyway there is no other way to have a longer path.

∴ option C is correct.

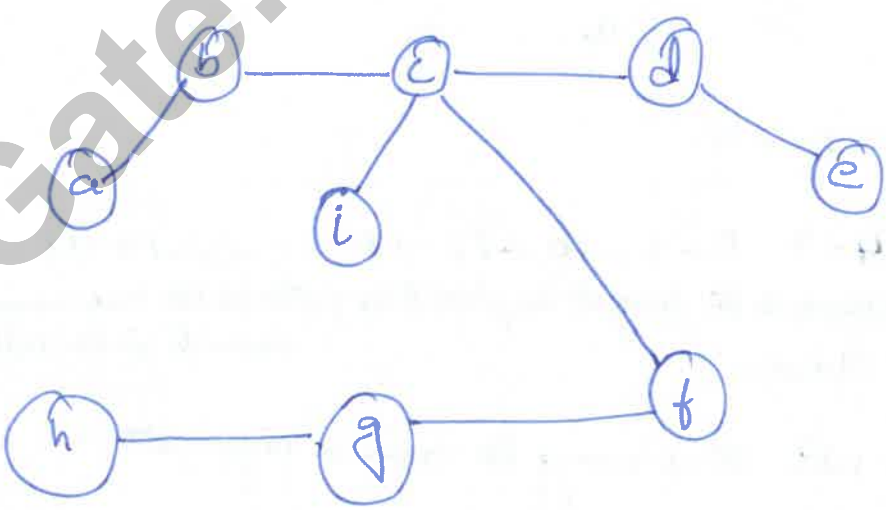
# Topic: Graphs: Spanning Trees.

## 61.1. Minimal Spanning trees :- What and Why ?




<u>Spanning Tree</u> A spanning tree is a tree which is constructed from the original graph G such that T.V = G.V (it has the same set of vertices) and T.E ⊆ G.V, (the edges are subset of the edges of the graph.).. The tree spans all the vertices of the graph.

An example spanning tree of the above graph is shown below.



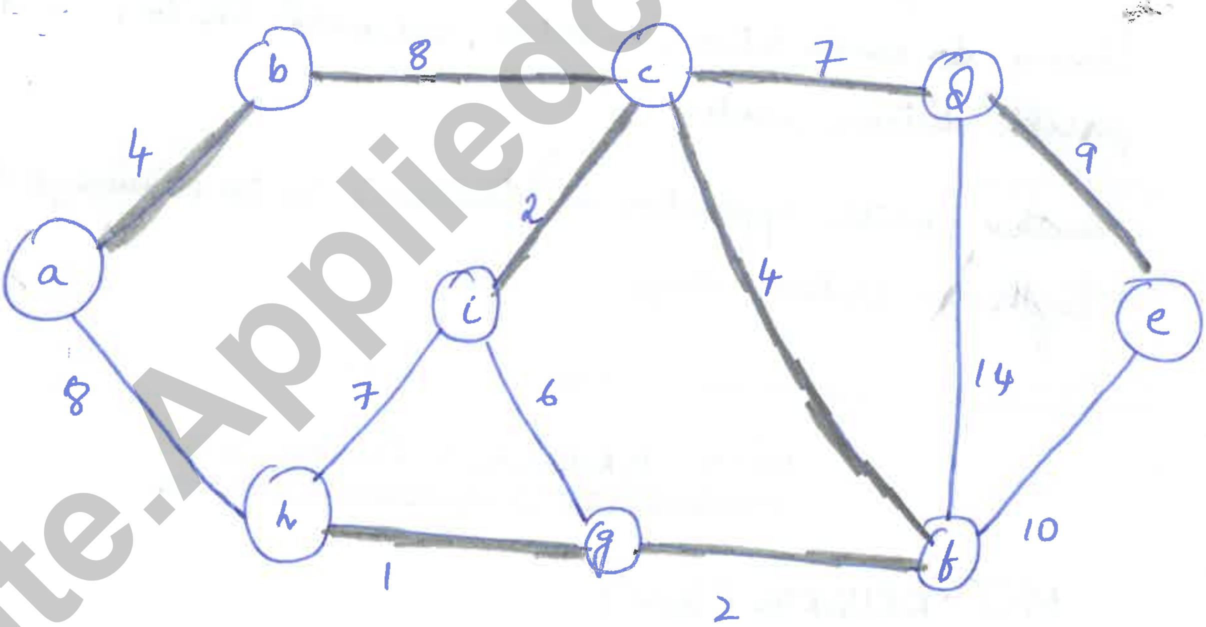

- A Tree is always an acyclic connected graph.

- A is the set of edges in the MST.
- Initially A is empty $\phi$. (line 1)
- Lines 2-3 we are creating a set for each vertex of the set.

1. {a} 2. {b} 3. {c} 4. {d} 5. {e} 6. {f} 7. {g}. 8. {h} 9. {i}

- Line 4 All the edges are sorted in non decreasing order of weight.
- Lines 5 to 8.

  the edges are taken in non decreasing order of weight and if the two endpoints belong to different sets then that edge is added to the MST and the two different sets are merged.




- 1. Smallest edge 1 (g-h)

  set 7 set 8 different sets

  Add to MST

  {g,h} is one set now. set 7.

**Minimal Spanning Tree :-** Given a spanning tree for a graph G.

- The cost of a spanning tree is the sum of cost of all the edges which are a part of the spanning tree.
- A Minimal spanning tree of a graph is that spanning tree which has minimum cost among all the spanning trees of that graph.
- There can be more than one minimal spanning tree for a given graph, but the cost of the MST will be the same.

- Why are MST's are important ?
- If the vertices are represent cities and we are representing the distance by the edge weight and if we are interested to lay down roads between the cities, we would like to have the minimal possible distance roads.
- Another possible application is Electronics is to minimize the wiring length in electronic chips.



## 61.2. KRUSKAL'S ALGORITHM

```
MST-KRUSKAL (G, w)
A = ∅
for each vertex v ∈ G.V
        MAKE SET(v)
sort the edges of G.E into nondecreasing order by weight w.
for each edge (u,v) ∈ G.E taken in non decreasing order by weight
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u,v)}
                UNION (u,v)
return A.
```

2. Next edge (c, i)
3 ≠ 9.

Add to MST. {c, i} is one set. set 3

3. Next edge (g, f).
7 ≠ 6.

Add to MST. {g, f} is one set. set 6.

4. Next edge (c, f).
set 3 ≠ set 6.

Now add c, f to MST
and the two sets are combined {c, i, f, g} − Set 3.

5. Next Edge (a, b)
1 ≠ 2

Add to MST. {a, b} is one set − Set 1

6. Next Edge (i, g)
3 = 3

− we cannot add i, g as both the ends belong to the same set.

7. Next Edge (i, h)
3 3

Cannot Add.

Ph: 844-844-0102

8. Next (c,d)

$3 \neq 4$

- Add c,d to MST
- Now {c,d,i,f,g,h} belong to one set - Set 3.

9. Next (b, c)

$1 \neq 3$

- Add (b,c) to MST
- Now {a,b,c,d,i,f,g,h} belong to one set Set 1

10. Next (a,h)

$1 = 1$

Cannot Add.

11. Next (d,e)

$1 \neq 5$

- Add (d,e) to MST
- Now {a,b,c,d,e,f,g,h,i} are one set - Set 1

12. Next (f,e)

$1 = 1$

Can't Add.

13. Next (d,f)

Cannot Add.

We have reached the end and have also constructed the MST.


Mail: Gatecse@appliedcourse.com

Cost of the MST = $4+8+7+9+2+1+2+4 = 37$.

- This is a Greedy Algorithm as at every step we are choosing the edges greedily. we are chosing the least cost edge remaining every time.

## Time Complexity

1. Lines 2-3 take $O(n)$ time
2. Line 4 :- Sorting takes $O(n \log n)$ time for m edges it takes $O(m \log m)$ time.
3. loop in lines 5-8 will take $O(m)$ time as it is executed for each edge and within the loop we are doing constant $O(1)$ time operations.

$\therefore$ Total time complexity $O(n) + O(m \log m) + O(m)$

$$= O(m \log m)$$

we know in the worst case $m = O(n^2)$

$$= O(m \log n^2)$$

$$= O(2m \log n)$$

$$= O(m \log n)$$

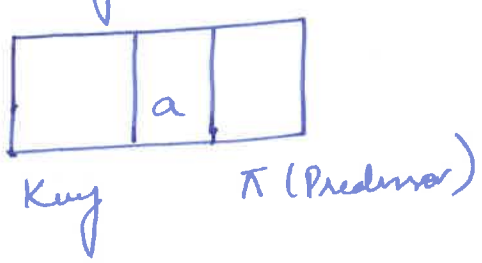
# 61.3 Prims Algorithm

MST-PRIM(G, w, r)

```
for-each u ∈ G.V
        u.key = ∞
        u.π = NIL
r.key = 0.
Q = G.V
while Q ≠ ∅
        u = Extract-Min(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u,v) < v.key.
                v.π = u
                v.key = w(u,v).
```

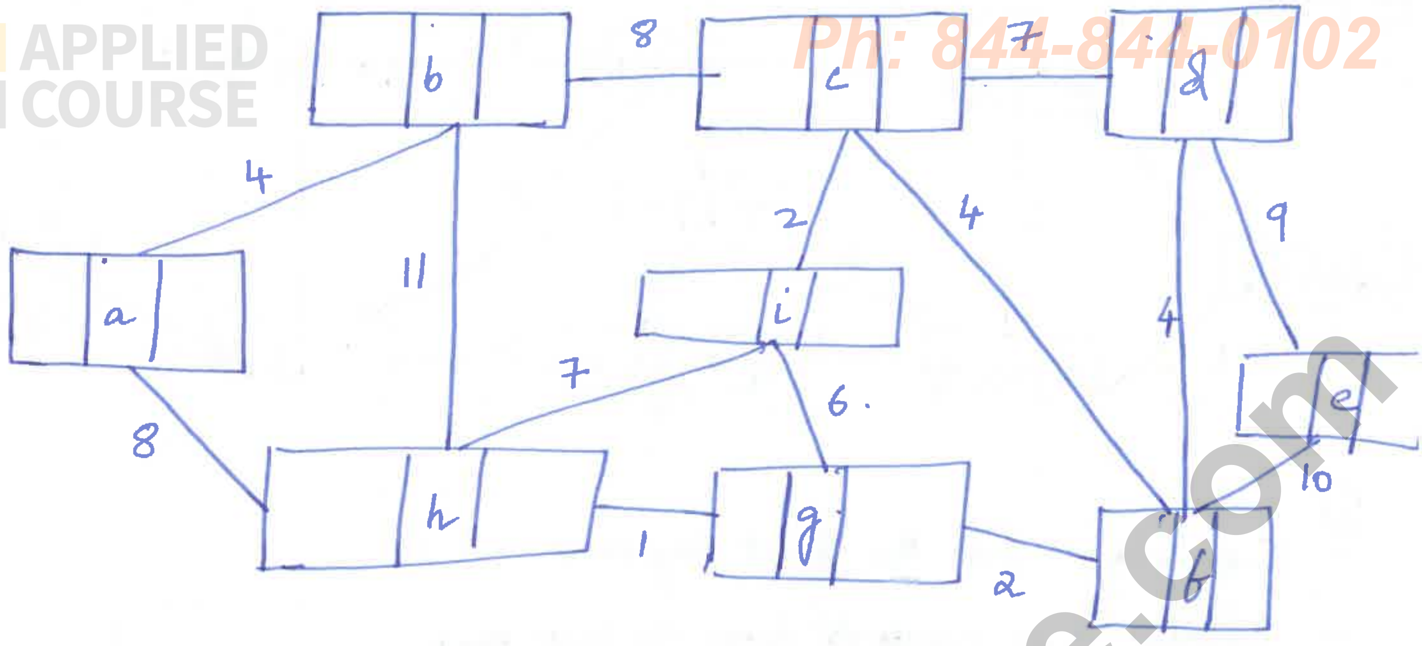
- Another way to find the MST for a graph

- The prims algorithm makes use of a data structure for each node/vertex where for each vertex the key and predessor are stored.




- The prims algorithm makes use of an Auxiliary data structure which is a min heap or/priority queue.

In the min heap




→ The above graph is same as the previous graph only difference is that it is represented using the two additional cells to store key and predecessor information.

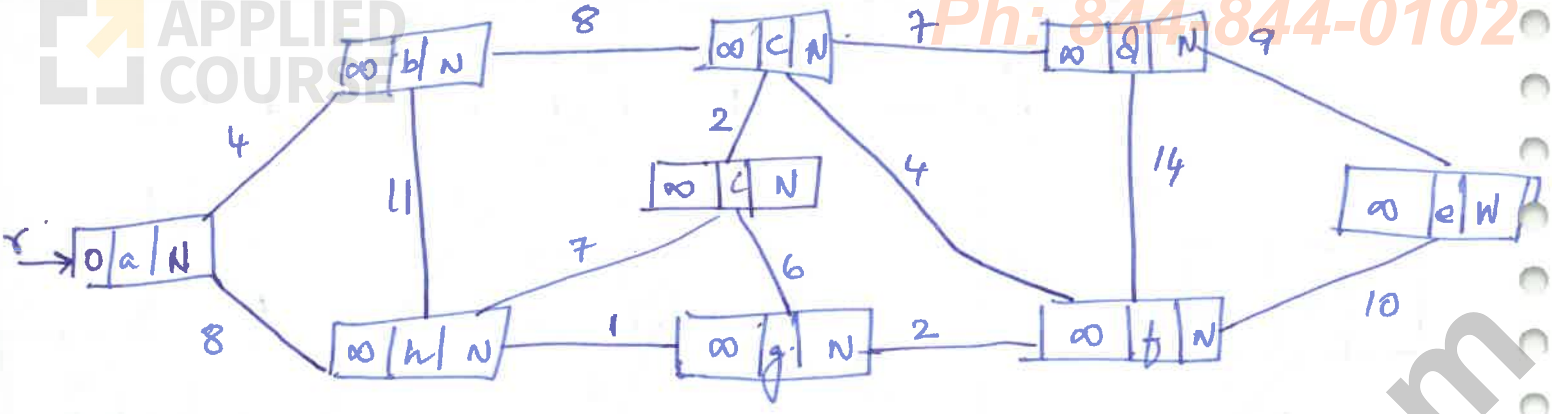
- 1. A min heap takes $O(\log n)$ time to extract the minimum element
  2. Min heap takes $O(\log n)$ time to modify the value of any element.

→ Lines 1–3 perform the initialization the key of every vertex is made $\infty$ and the ~~distance~~ predecessor of every node is made NIL.

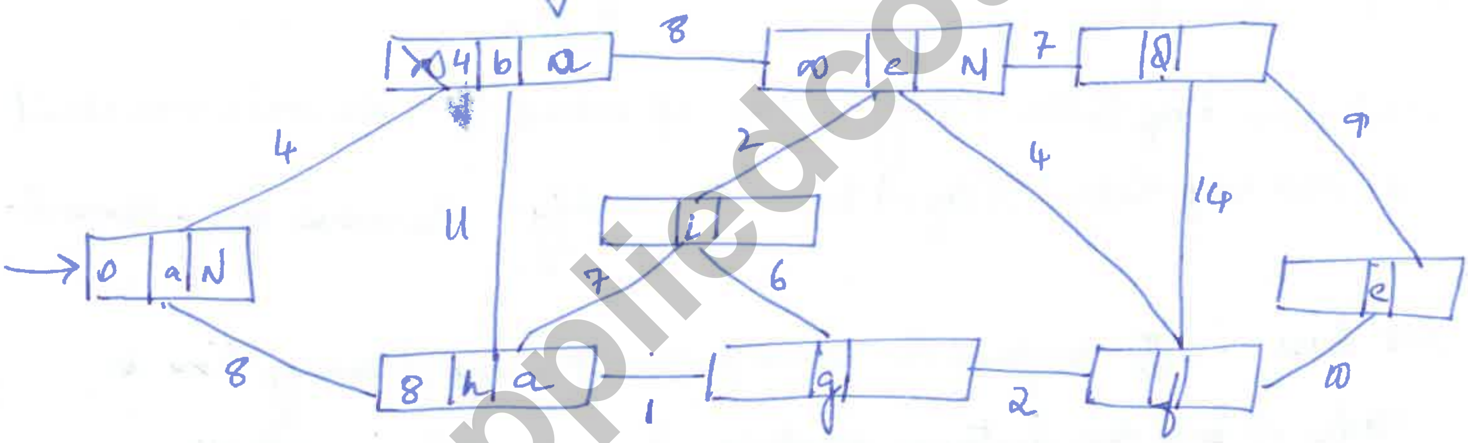
→ Line 4 assigns the key of the source vertex as 0.

- Line 5 adds all the vertices to the min heap.

- From lines 6–11 at each iteration of the loop we extract the element with the minimum key value and its neighbouring vertices are explored, if a path is found with lesser key value then the key is updated and the predecessor is also updated.



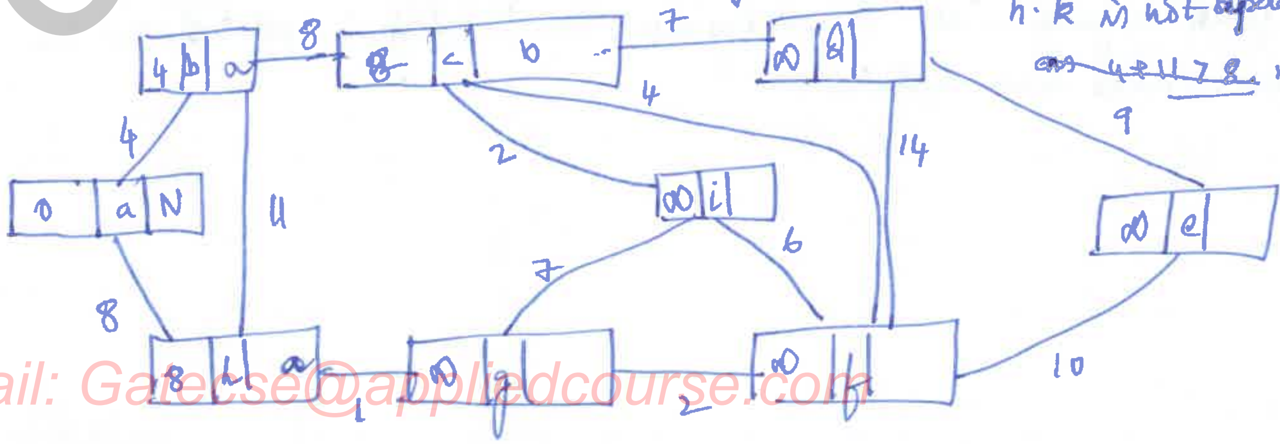
(1)
- From all a has the least key value
- Initially a is removed from the min heap.
- Its neighbours b and h are explored and are updated to 0+4=4 and 0+8 = 8 respectively.



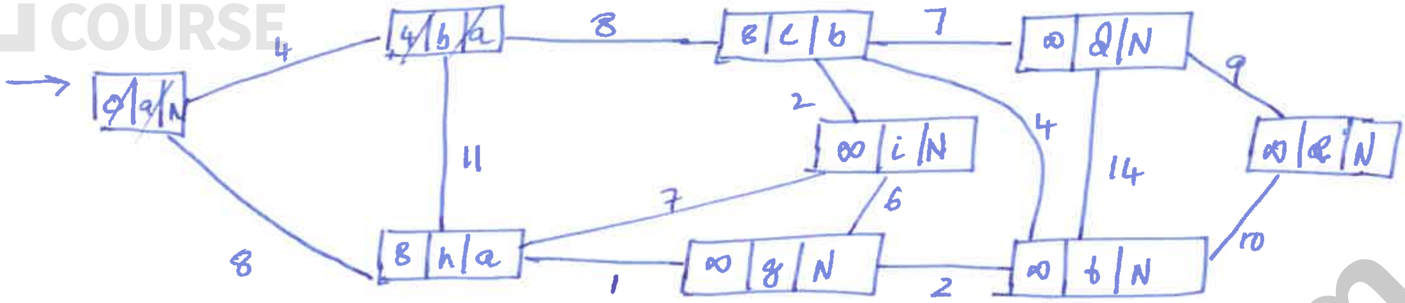
(2)
Now the least possible node with least key value is 4 node b
- b is removed from the priority queue.
- Its neighbours are c, h they are updated c.k = 8 = 12

h.k is not updated

~~as~~ 4+11 ≮ 8 ×

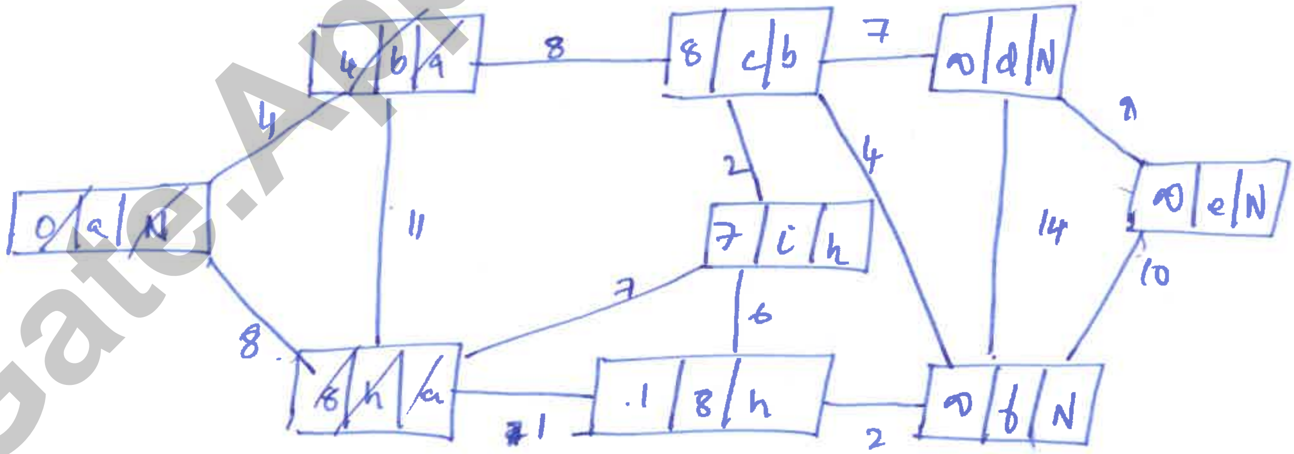


③ Now we can choose either h or c. lets choose node h.




- Nodes which are neighbouring are a, b, i, g.

1. a is already removed from the Q
2. b is also removed from the Q.
3. i is updated as 7.
4. g is updated as 1.



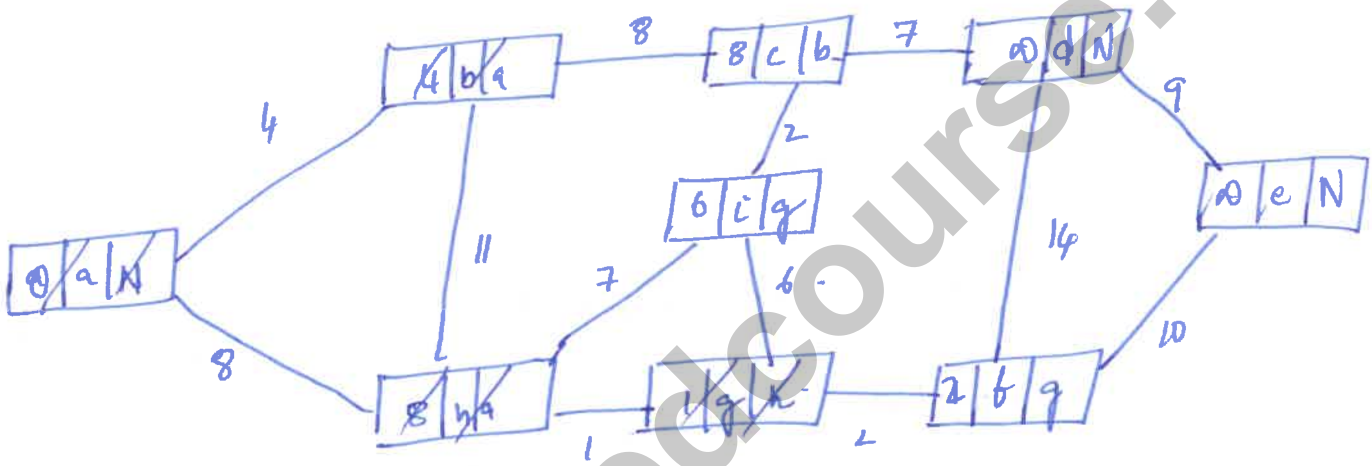
(4) Now g has minimum key value, it is removed from the Q.

Nodes which are neighbouring are h, i, f.

1. h is already removed from the Q.

2. i is updated as 6.

3. f is updated as 2.



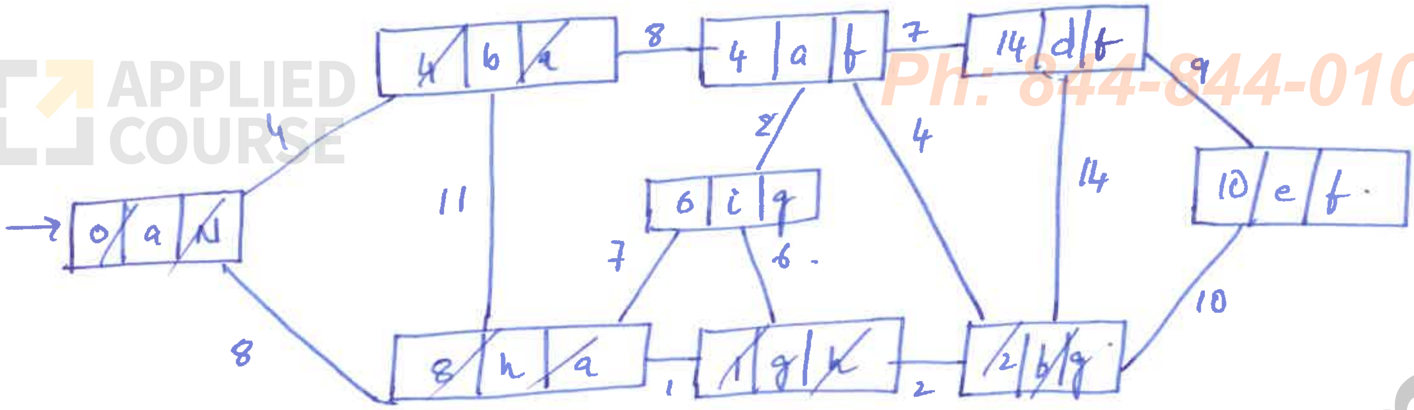
(5) Now node f has minimum key value, it is removed from the Q. Neighbouring nodes are c, d, e, g.

1. g is already out of the Q.

2. c is updated to 4

3. d is updated to 14

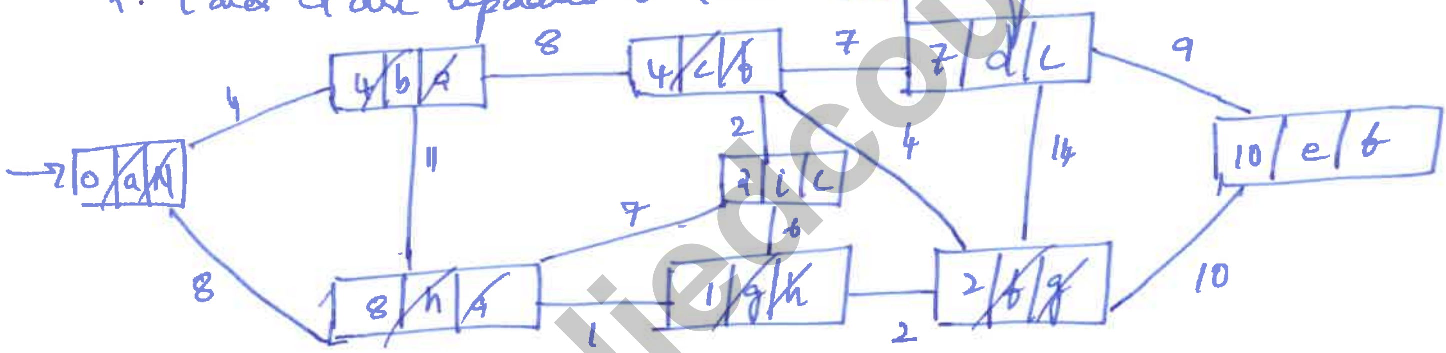
4 e is updated to 10.



(6) Now node c is removed from the Q

Its neighbours are i, f, b, d.

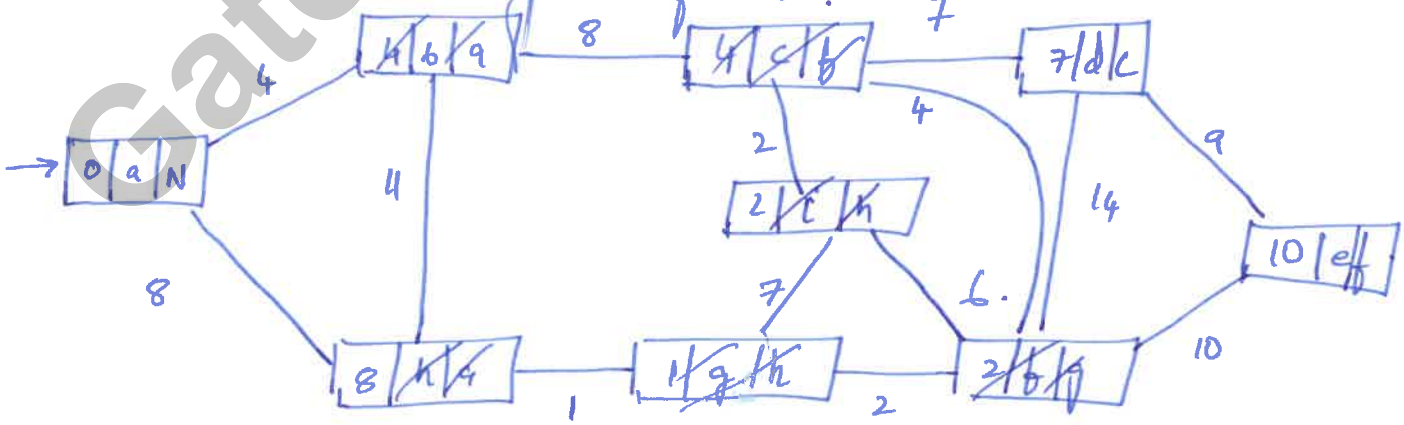
1. b and f are already out of the Q.
2. i and d are updated to 2 and 7 respectively.



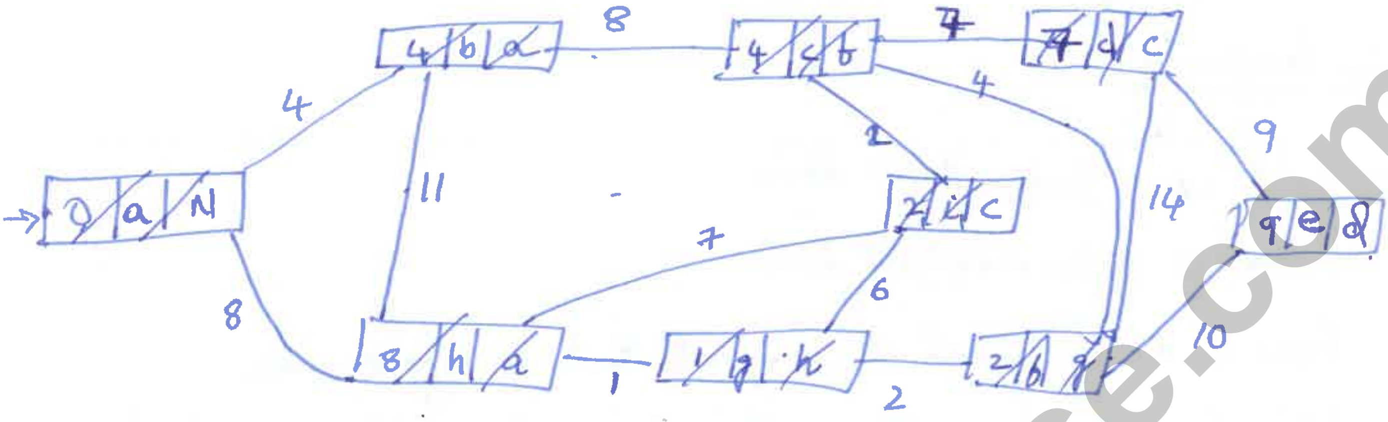
(7) Now node i is removed from the Q.

It has neighbours h, c, g.

1. All are already out of the Q.

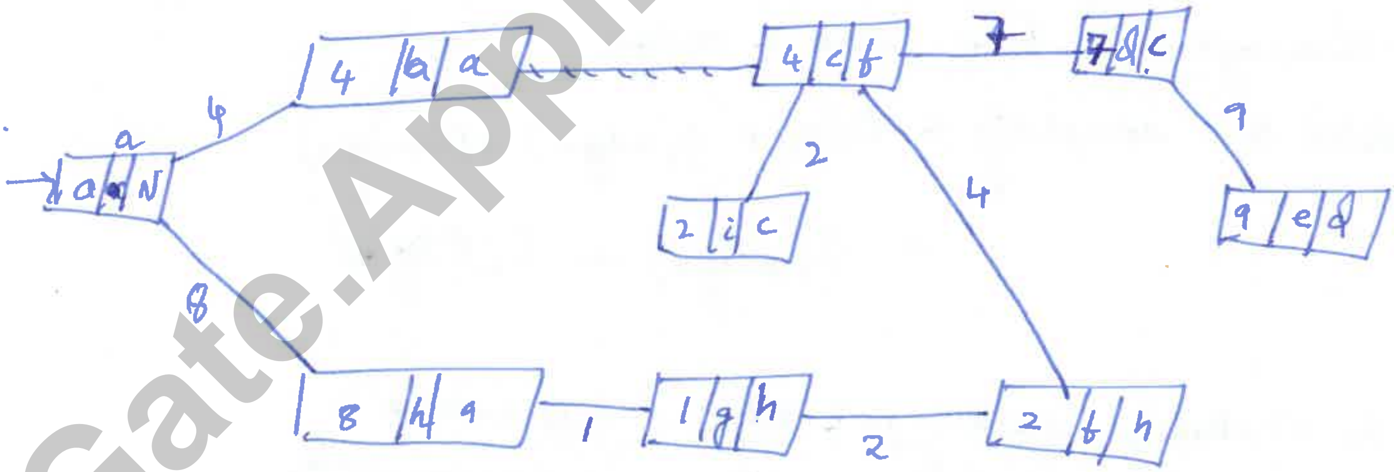


(8) Now node d is removed from the Q
It has neighbours c, f, e
1. c, f are already out of the Q.
2. e is updated to 9.




(9) Now e is removed from the Q
It has neighbours d, f but both are already out of the Q.

The MST can be traced back using the predecessor info in each node.
following is the MST.




The cost of the MST is given by = 4 + 7 + 9 + 2 + 4 + 2 + 1 + 8
= 37

- At every iteration we are finding minimum distance vertex at every step

- Here at every:

Time Complexity

- Lines 1 to 3 takes $O(n)$ time
- Lines 4 takes constant time
- Line 5 to construct the heap will take heapify operation
- Line 6-11 the loop will execute n times. ($n$ = no of ~~edges~~ vertices)
  - line 7 for extracting min it takes $\log n$ time for n

- the loop 8-11 will execute m times (no of edges = m).
  - line 11 modification/updation will take $\log n$ time

time complexity of line 7 $n \log n$.

time complexity of line 10-11 $= m \log n$

Total time complexity $= O(n) + O(n \log n) + O(m \log n)$

$= O(m \log n)$ or $O(E \log V)$

- In Kruskals we greedily pick edges - sets are used
- In Prims we greedily pick vertices. - vertices are used.

# 61.4 Properties Of MST

① G(V,E) is a graph, a spanning tree of the graph G will always consist of (|V|-1) edges.

(E−V+1) edges are not part of the spanning tree

② If we add any new edge to a cycle spanning tree it becomes cyclic. Any spanning tree is maximally acyclic.

③ Every spanning tree is minimally connected, if any edge is removed it will disconnect the graph.

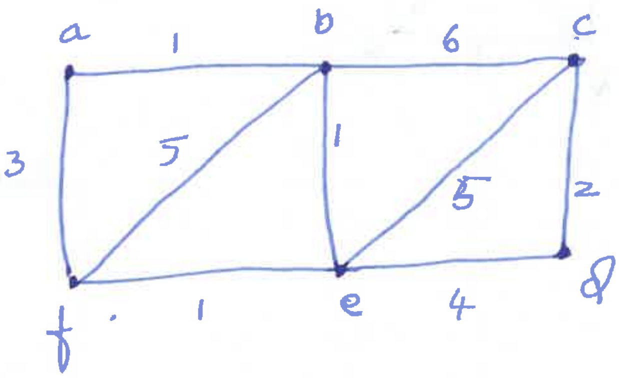
④ There may be several MST of same weight of a given graph.

⑤ If each edge has a unique/distinct weight then you will have exactly one unique MST. Uniqueness Property.

⑥ Cycle Property :- For any cycle C in graph, if the edge weight (e) is larger than all other edges in C, then edge(e) cannot belong to MST.

⑦ Cut-Property

If we delete edges {bc, ec, ef}
the graph will be divided into
{a,b,d,e} and {c,f}



For any cut C, if the weight of an edge in the cut set is strictly smaller than all other weights of edges in the cut set then this edge must be part of MST.

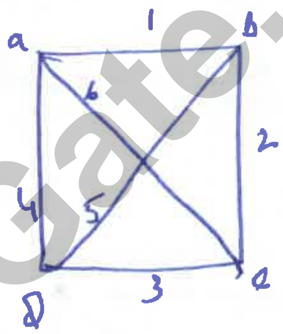
- In the above example of (bc, ec, ef) the smallest is ef is part of MST always.
  6, 5, 4

(8) Minimal Cost Edge :- If the minimal cost edge in a graph is unique then it must always belong to the MST.

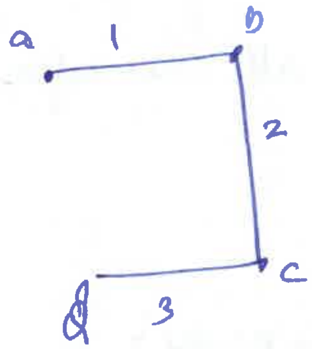
---

61.5 Solved Problem GATE 2018

Let G be a complete undirected graph G on 4 vertices having 6 edges with weights 1, 2, 3, 4, 5 and 6. The maximum possible weight that a minimal spanning tree can have is ____.



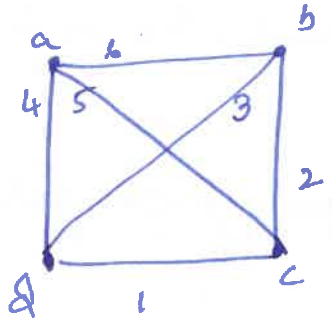

MST



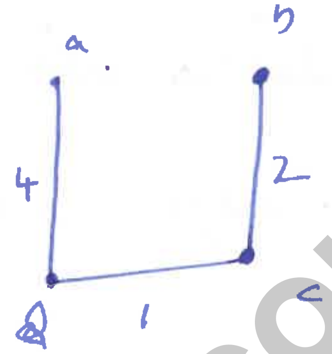

Weight = 1+2+3 = 6.

Can we get a spanning tree of Wt 8?

We assign the 3 largest weights to given single vertex

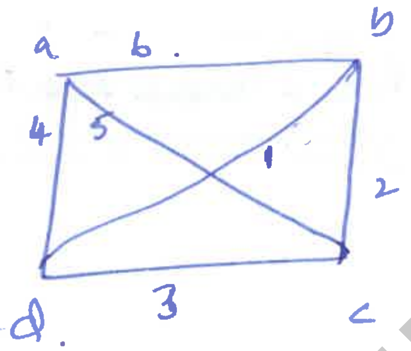



Now MST

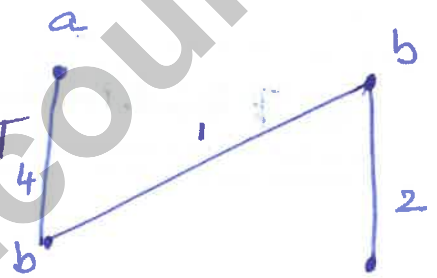



$$\text{wt}(MST(4)) = 4+1+2 = 7$$

If we try to make the diagonal as wt 1 least wt




MST




$$\text{Wt of MST} = 4+1+2 = 7$$

Whatever combination we try we cannot get more than 7.

Answer is 7.

---

## 61.6 Solved Problem-2

Q) What is the maximum no of undirected graphs with n-vertices.?

- A Graph of n vertices → can have a maximum of $\frac{n(n-1)}{2} = m$ edges.

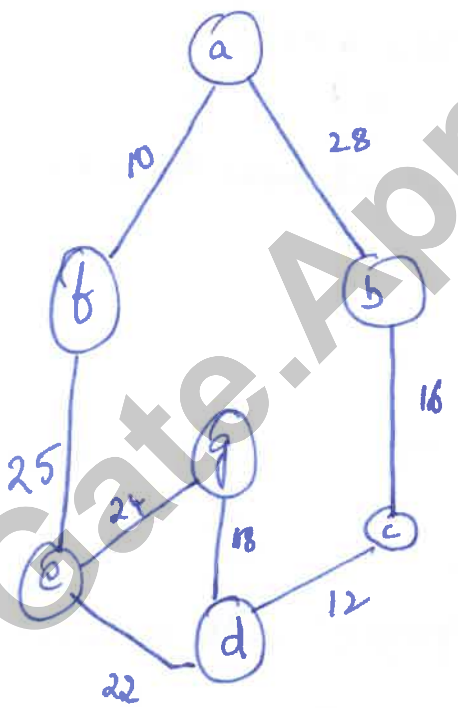
or between any two pair of vertices if we consider an edge then we get the above $\frac{n(n-1)}{2}$ edges.

- In a graph the no of different graphs we can construct by either including or excluding a particular edge, each edge has 2 possibilities either it is present or absent, this is for each of the $\frac{n(n-1)}{2}$ edges. ∴ The total no of unique graphs possible is

$$2 \times 2 \times \cdots \cdots 2 \quad \frac{(n(n-1))}{2} \text{ times}$$

$$= 2^{\frac{n(n-1)}{2}} \text{ graphs}$$

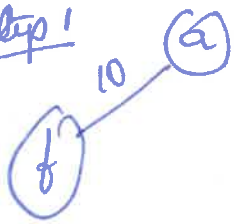
## 61.7 Solved Problem: MST using Prim's & Kruskal's Algorithm



On applying Kruskal's algorithm

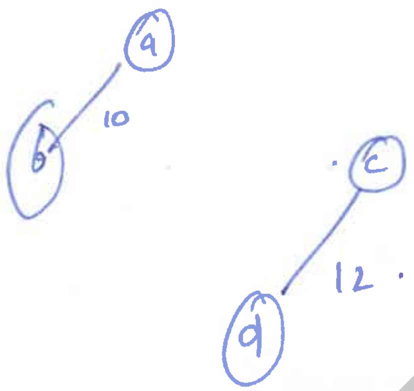
1. {a} 2. {b} 3. {c} 4. {d}. 5{e} 6. {f} 7. {g}

Step 1



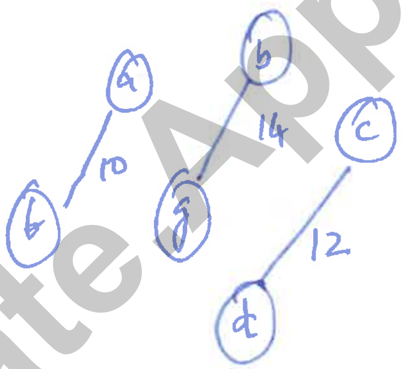

1 {a,f} 2{b} 3{c} 4{d} 5{e} 6{g}

Step 2



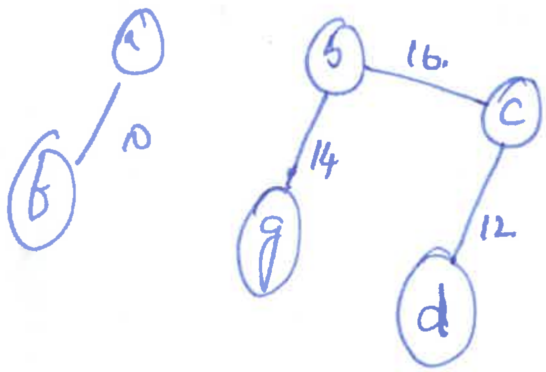

1. {a,f} 2. {b} 3. {c,d}
6. {e} 6. {g}.

Step 3




1. {a,f} 2{g,b} 3. {c,d}
4. {e}.

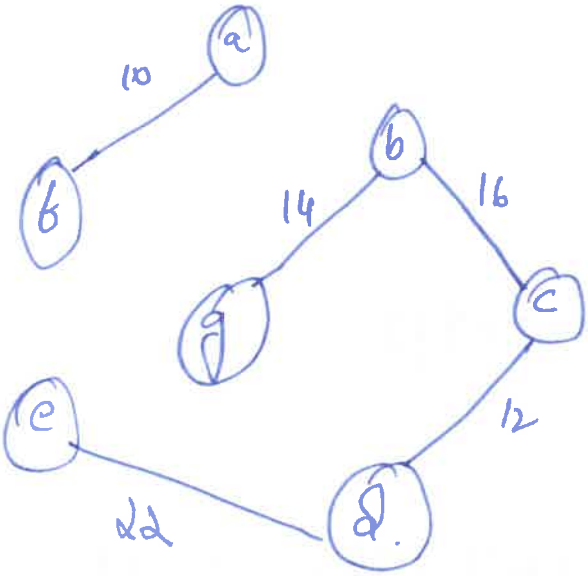
Step 4




1. {a,f} 2{b,c,d,g}
4. e.

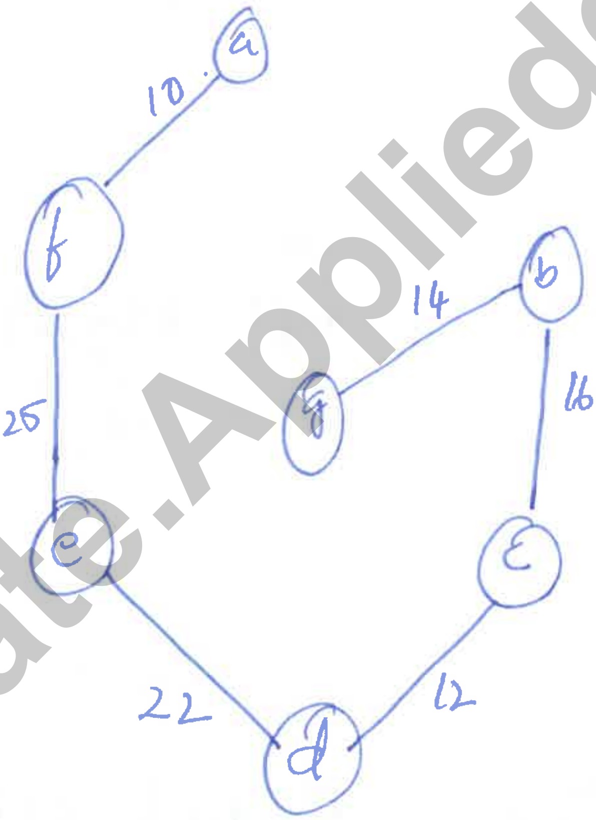
Step 5: g d cannot be added

Step 6

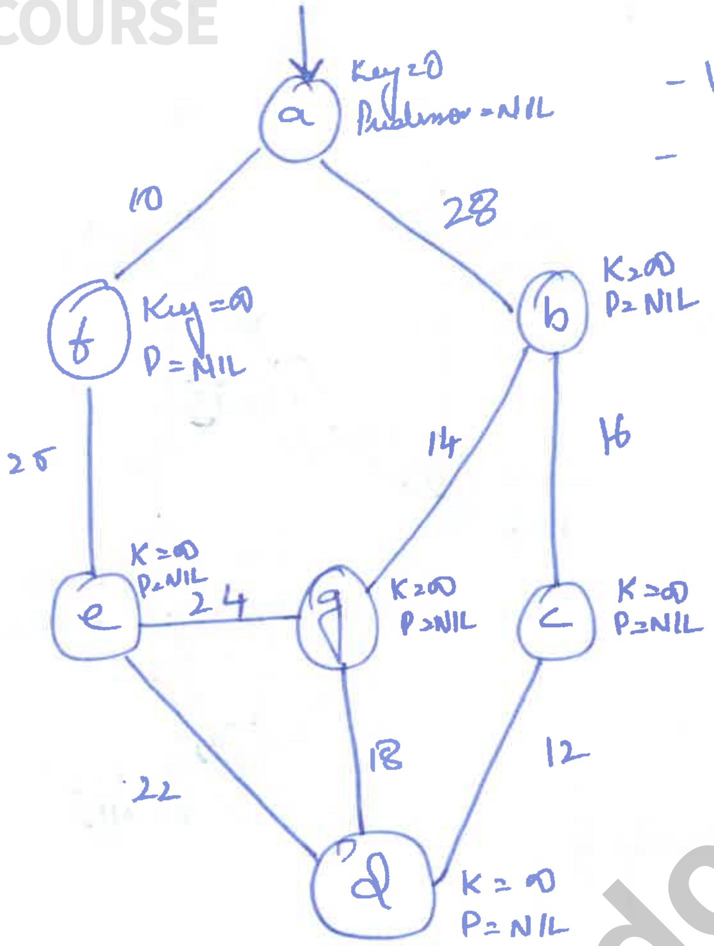



1. {a, f}
2. {b, c, d, e, g}.

Step 7.



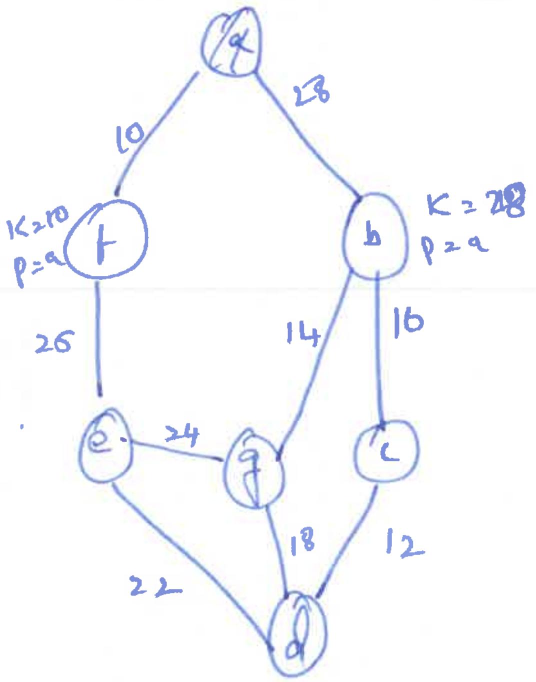
Lets Apply Prims Algorithm.




- We make use of priority Q in Prims
- Key and predessor info is recorded.

Step 1 A is removed from PQ.

f. Key = 10
f. P = a
b. k = 28
b. P = a.



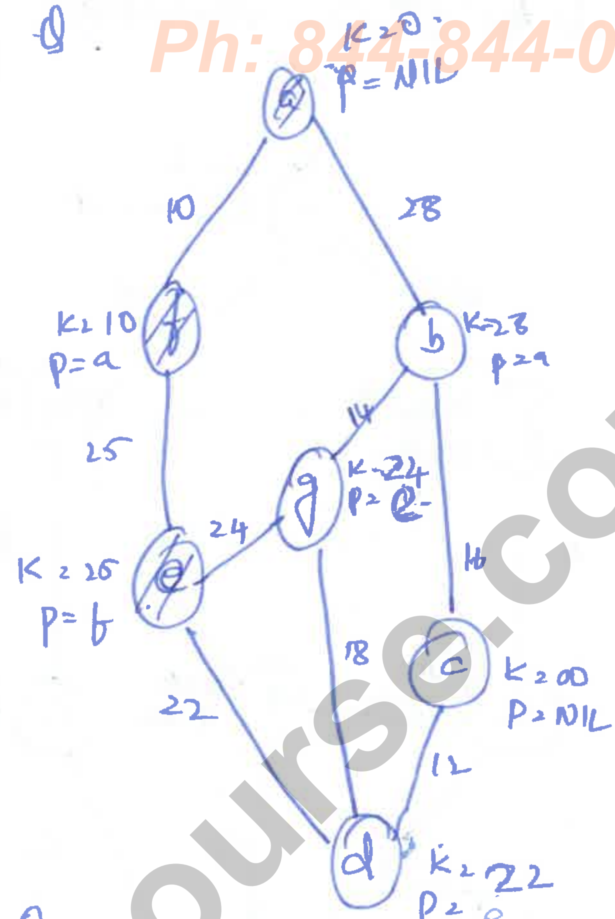

Step 2 f is removed from Q

e.K = 25
e.P = f

③ e is removed from the priority Q.

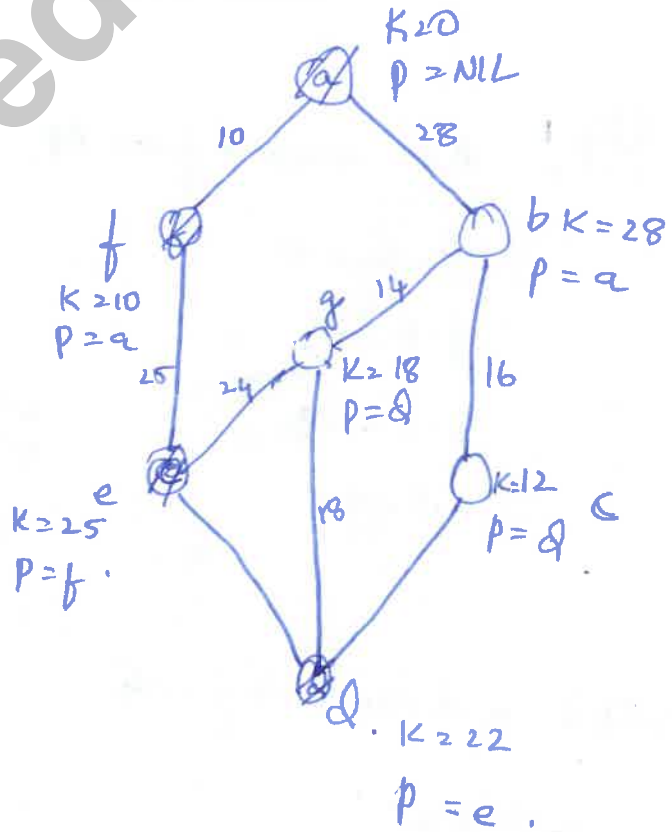
g.K = 24
g.p = e.

d.K = 22
d.p = e




④ d is removed from the priority Q

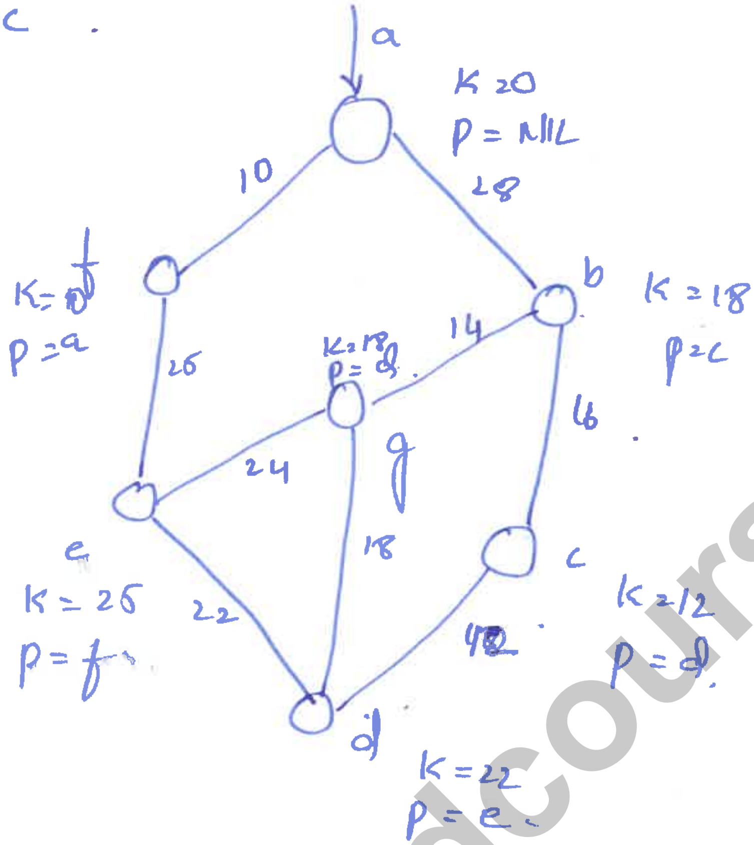
c.K =12 , c.p = d
g.K = 18 , g.p=d.



⑤ c is removed from the priority Q.

b.K = 16

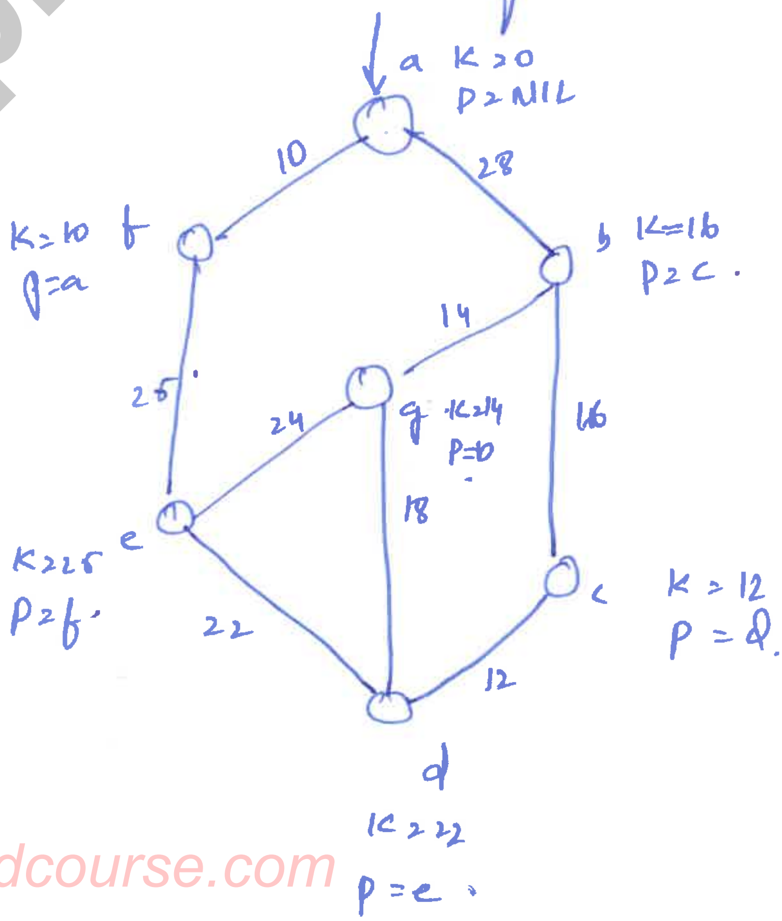
b.p = c.



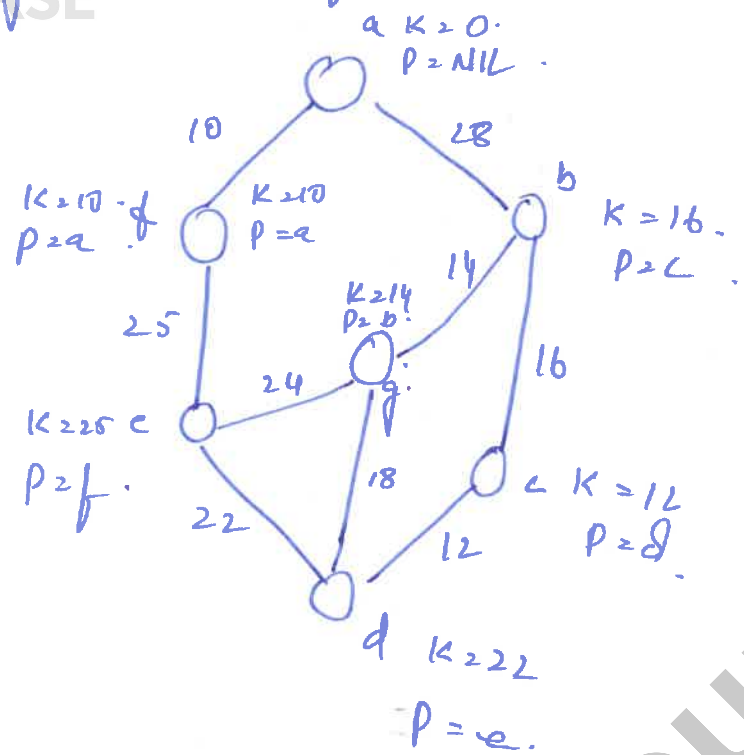

⑥ Now b is removed from the priority Q.

g.K = 14

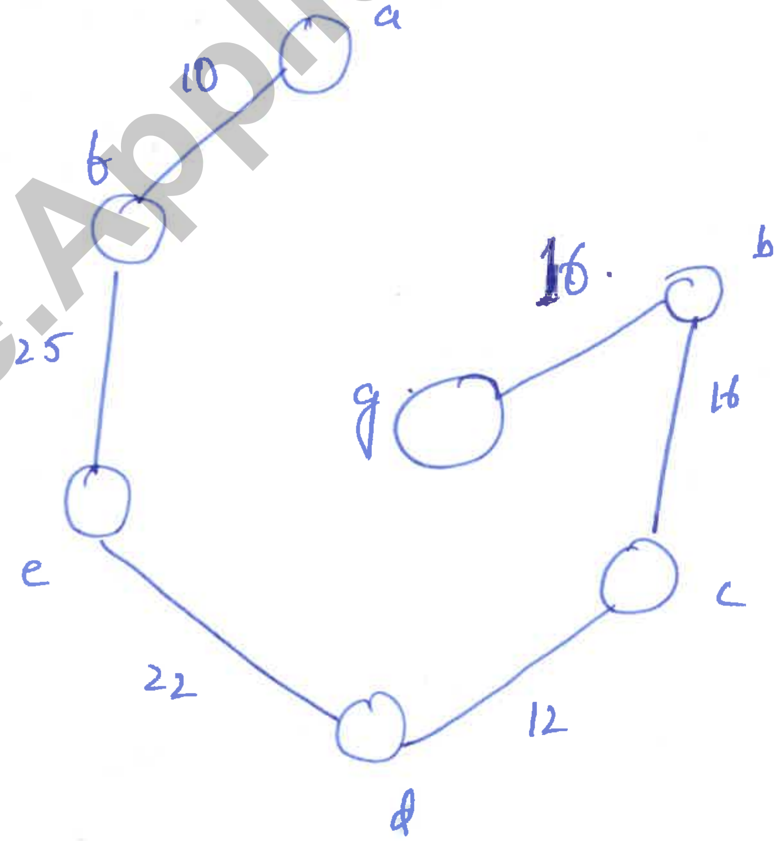
g.p = b.



(7) Now g is removed from the queue.




We can trace the MST by using the predecessor info of every node



# 61.8 Solved Problem



## GATE 2016

→ G = (V, E) is an undirected simple graph in which each edge is of distinct weight and e is a particular edge of G. Which of the following statements about the MST of G is/are TRUE.

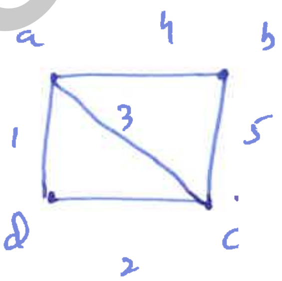
I. If e is the lightest edge of some cycle in G, then every MST of G includes e.

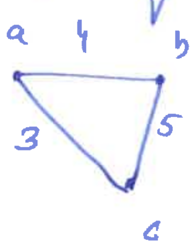
II. If e is the heaviest edge of some cycle in G, then every MST of G excludes e.

A. I only
B. II only
C. both I and II
D. neither I and II.

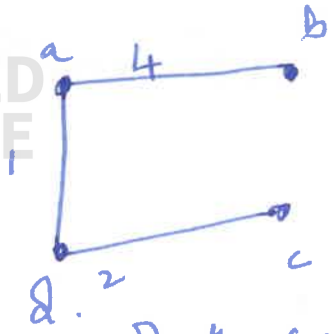
Statement II is true from the cycle property of the MST.

For statement I let us consider the following example graph.




In the cycle a be the lighter of edge is ac but if we construct the MST for it we will get




Here 3 is part of the cycle abc but it is not part of the MST.

statement I is not true.

## 61.9 Solved Problem GATE 2005

Q) An undirected graph G has n nodes. Its adjacency matrix is given by an n×n square matrix whose

(i) diagonal elements are 0's and.

(ii) non-diagonal elements are 1's.

Which of the following is TRUE?

(A) Graph G has no MST

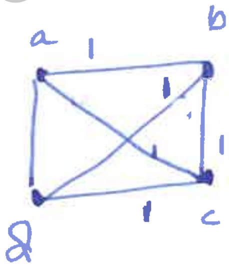
(B) Graph G has a unique MST of cost (n−1)?

(C) Graph G has multiple distinct MST's each of cost n−1

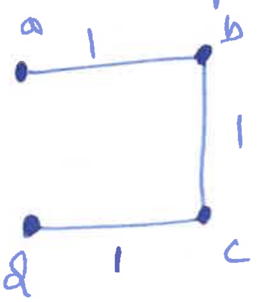
(D) Graph G has multiple spanning trees of different costs.

Let us consider n=4

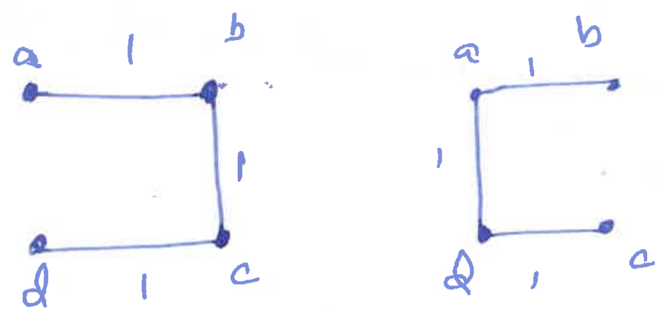
$$\begin{array}{c c} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$




option A can be eliminated because a MST always exists following is an example.



(B) option B can also be eliminated because G has more than one spanning tree of the same cost




(C) for option C is correct because the above two examples justify it.

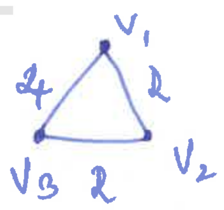
(D). Graph G has multiple spanning tree of multiple costs, this is not possible because the MST will always contains (n-1) edges. and cost of which will add upto (n-1).

## 61.10 Solved Problem GATE 2006.

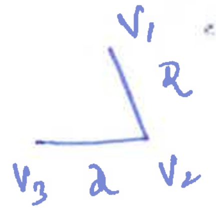
Consider a weighted complete graph G on the vertex set $\{v_1, v_2, \cdots v_n\}$ such that the weight of the edge $(v_i, v_j)$ is $2|i-j|$. The weight of the MST of G is:-

(A) $n-1$
(B) $2n-2$
(C) $^nC_2$
(D) 2

let us consider graph with $n = 3$




Here MST




wt of MST $= 2+2$
$= \underline{4}$

We can eliminate option D

Option A $- \; 3 - 1 = 2$ can be eliminated

Option B $\; 2n-2 = 2\times3-2$

$= 4$ is correct - but let us check.

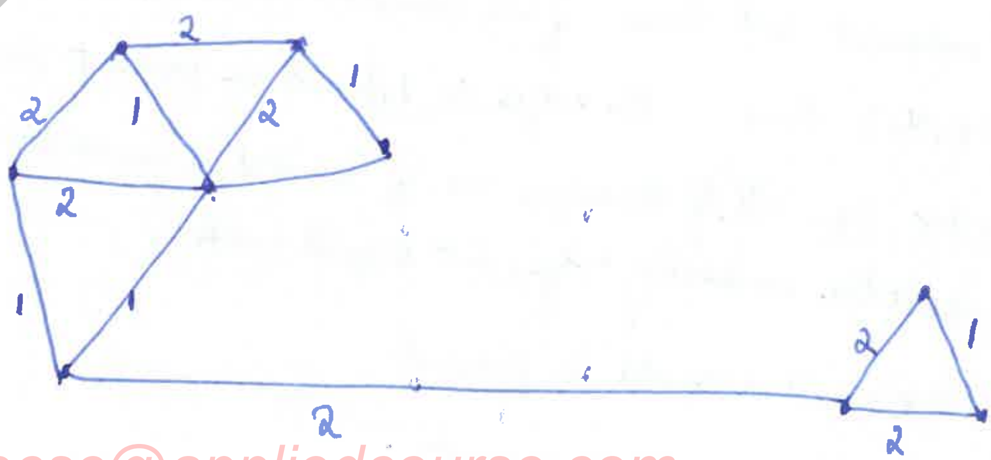
Option C $\; {}^{n}C_2 = {}^{3}C_2 = 3.$ is not correct.

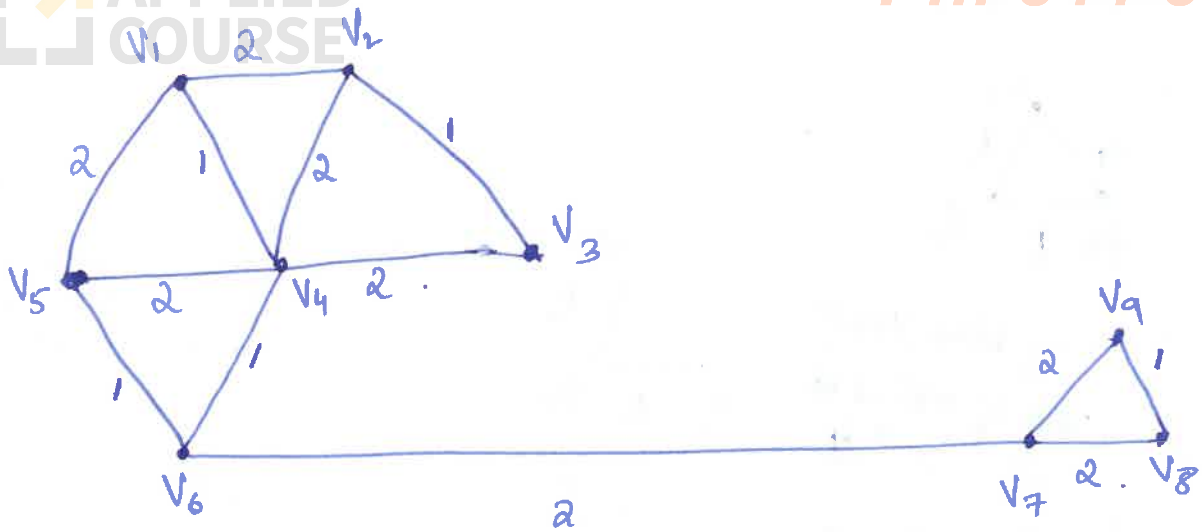
<u>option B</u> is the correct Answer.

---

## 61.11 Solved Problem GATE 2014

The no of distinct minimal spanning trees for the weighted graph given below is ______.

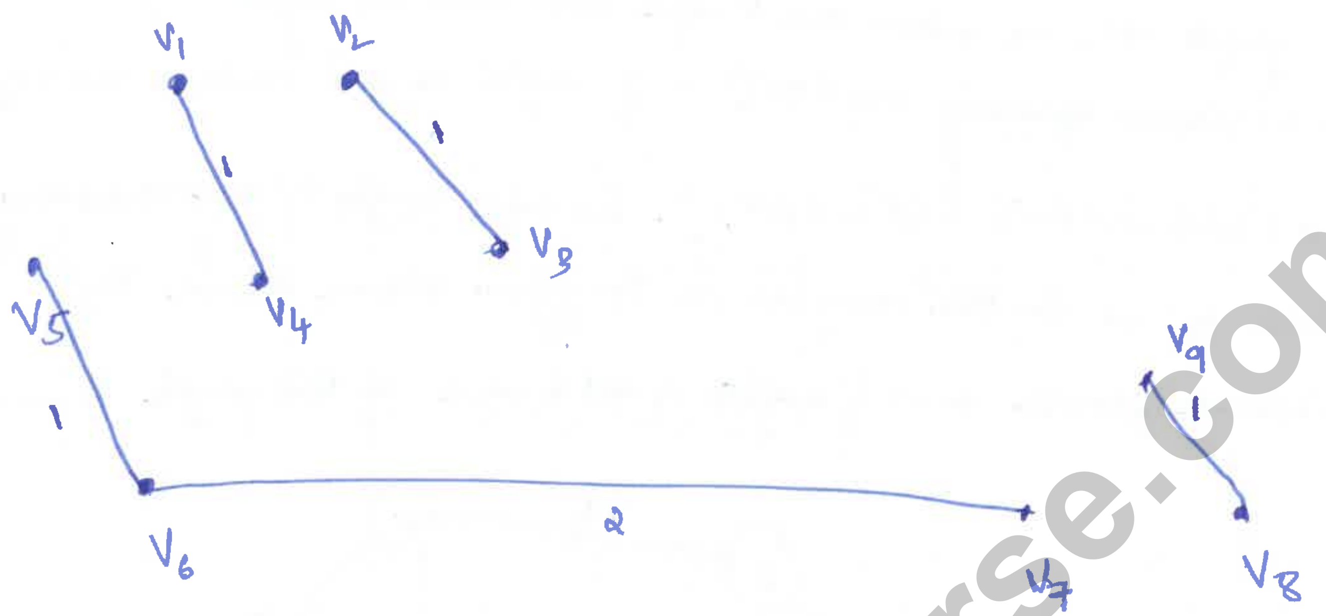




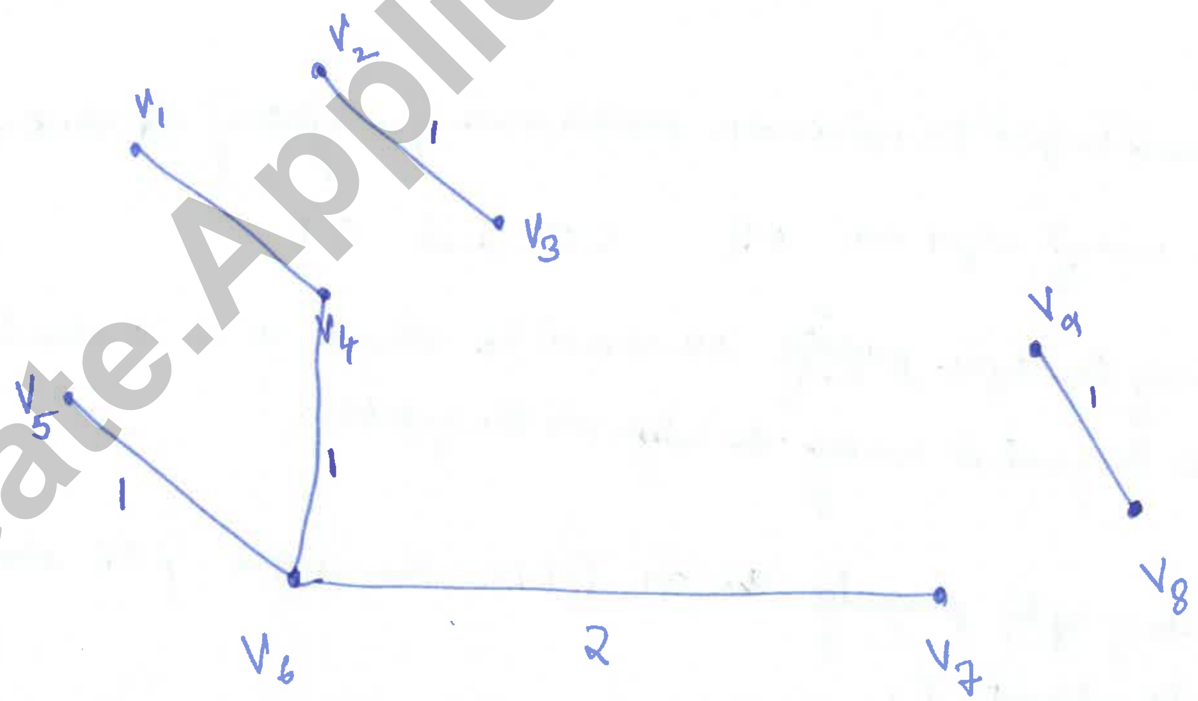

By using the cut property of MST let us determine the edges which are mandatory to be present in the MST

1. If we consider vertex $V_1$ from other vertices $(V_1, V_4)$ edge must be present
2. If we consider $V_2$ $(V_1, V_3)$ must be present
3. If we consider $V_3$ $(V_2, V_3)$ must be present
4. If we consider $V_4$ $(V_1, V_4)$ or $(V_4, V_6)$ must be present it is confusing lets keep it aside for some time
5. for $V_5$ $(V_5, V_6)$ must be present.
6. for $V_6$ we consider $(V_5, V_6)$, $(V_4, V_6)$ must be present let us keep it aside for the moment but then if we consider the cut $(V_1, V_2, V_3, V_4, V_5, V_6)$ and $(V_7, V_8, V_9)$ then the edge $(V_6, V_7)$ must be present in the MST
7. If we consider $V_7$ all of the edges are of weight 2 we cannot say for sure about the mandatory edge, lets keep it aside.
8. for $V_8$ $(V_8, V_9)$ should be present
9. for $V_9$ $(V_8, V_9)$ should be present

Lets draw the graph with mandatory edges for the MST




Now if we consider the cut $(V_1\ V_2\ V_3\ V_4)$ and $(V_5\ V_6\ V_7\ V_8\ V_9)$ it has the connecting edges $(V_1 V_5)$ $(V_5 V_4)$ $(V_4 V_6)$ & of which $V_4 V_6$ should be present adding it we get the below graph.




Now in between the ~~order~~ vertices $(V_2, V_3)$ and $(V_1 V_4 V_5 \cdots V_9)$ we have 3 choices each of wt 2.

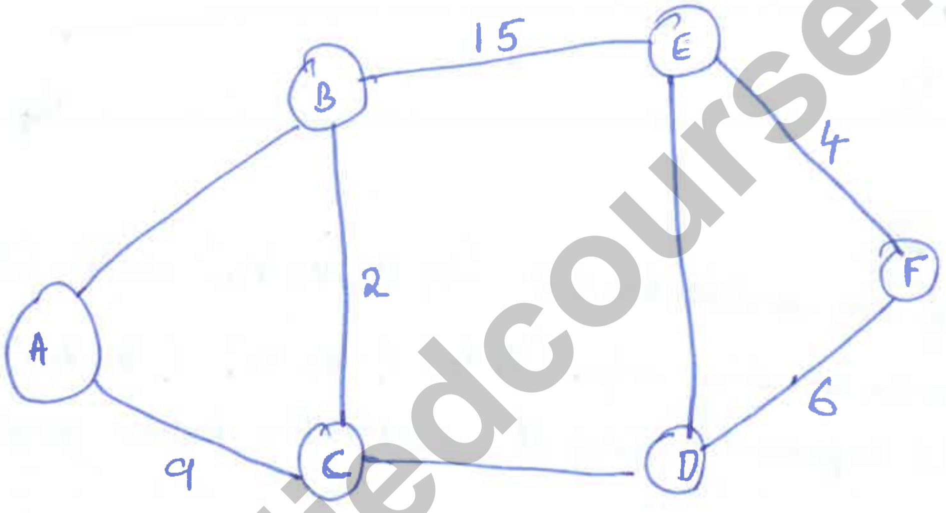
In between the vertices $(V_1 V_2 \ldots V_7)$ and $(V_8 V_9)$ we have 2 choices.

Total no of ways are $3 \times 2 = \underline{6}$

# 61. 12. Solved Problem Gate 2015



→ The graph shown below has 8 edges with distinct edge weights. The minimum spanning tree (MST) is of weight 36 and contains the edges {(A,C), (B,C), (B,E), (E,F), (D,F)}. The edge weights of only those edges which are in the MST are given in the figure shown below. The minimum possible sum of weights of all 8 edges of this graph is ____.




We need to find the minimum possible sum of weights of all the 8 edges

The missing edges are AB CD and ED.

- Using the cycle property AB should be atleast 10 as it should be the greatest among the edges in the cycle ABC.

- Using cycle property on EFD the weight of ED should be atleast 7.

- Using cycle property on BEDC the weight of CD should be atleast 16.

Total weight should be atleast = 36 + 10 + 7 + 16

= 46 + 23 = $\underline{\underline{69}}$

# TOPIC GRAPHS: SHORTEST PATHS

## 62.1 Shortest paths: What and Why?

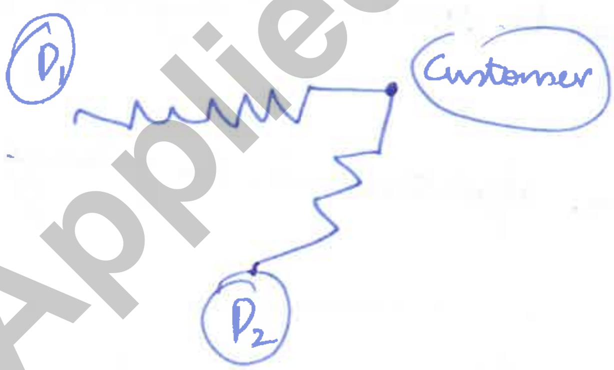
### Some applications

1. Google Maps:


(routing problems)

Finding the shortest distance from given source to destination u to V.

Uber has to recommend the closest set of drivers for a cab request




Another example is the internet graph in computer networks where each node is . . . and the edge is the link in between them.

when there is a request on the internet we need to find the best possible route to for the server.

- Shortest path is very widely used in real world and has many applications.

- Shortest paths are of 2 variants.

(1) Single Source shortest path : To find the shortest path from one vertex to all other vertices in the graph

(2) All pairs shortest paths :- To find the shortest path b/w all pairs of vertices of the graph.

Here we have two variations possible.

(1) All edge weights are positive eg road network

(2) Edge weights may be positive or negative example money transfer etc

for all variants we have algorithms in the following videos.

# 62.2. DIJIKSTRA's ALGORITHM

- It is for the Single Source Shortest path.

*** → Works properly only if all the edge weights are non negative

DIJKSTRA(G, w, s)

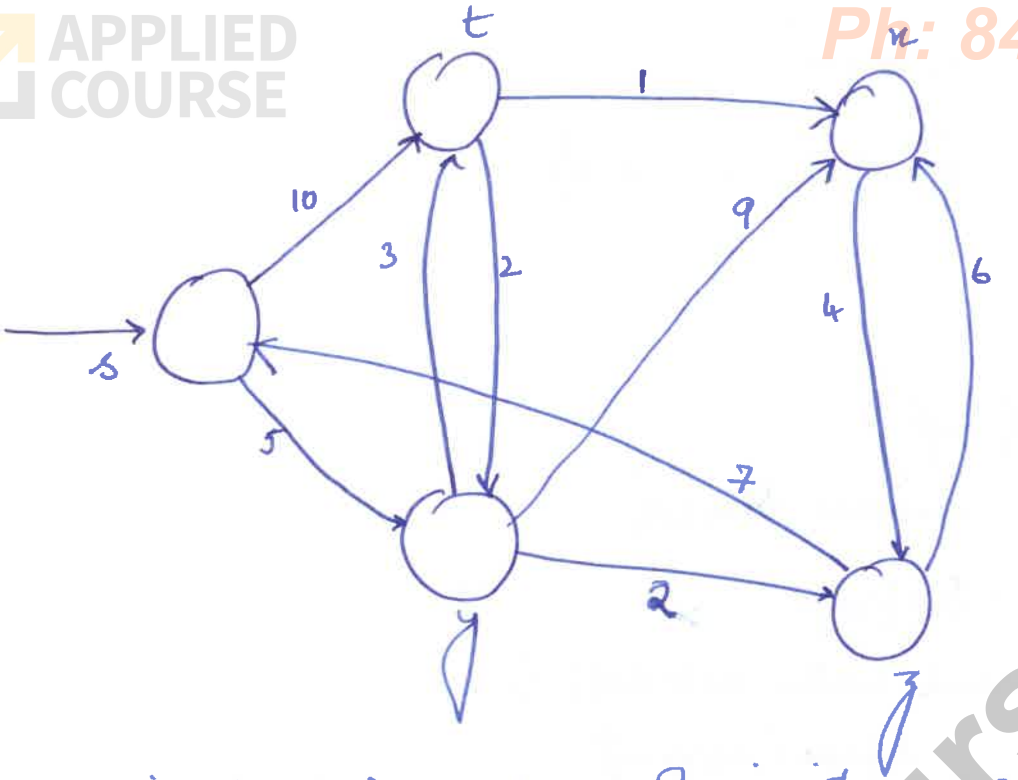
1. INITIALIZE-SINGLE-SOURCE(G, s)
2. $S = \phi$
3. $Q = G.V$
4. while $Q \neq \phi$
5. $u$ = EXTRACT-MIN(Q)
6. $S = S \cup \{u\}$
7. foreach vertex $v \in G.Adj[u]$
8. RELAX(u, v, w)

RELAX(u, v, w)

1. if $v.d > u.d + w(u,v)$
2. $v.d = u.d + w(u,v)$
3. $v.\pi = u$.

INITIALIZE-SINGLE-SOURCE(G, s)

1. for each vertex $v \in G.V$
2. $v.d = \infty$
3. $v.\pi = NIL$
4. $s.d = 0$.




→ The Dijkstra's algorithm makes use of a priority queue.

→ For each node as in the prims algorithm we have distance and predessor information

→ ~~One~~ line 1 is for intalizing all the vertices, the distance is intalised to ∞ and the parent to N except for the source for the source vertex the distance is intalized to 0.

→ On line 2. the set S is the set of vertices for which the shortest path has been computed is intalized to null set ∅.

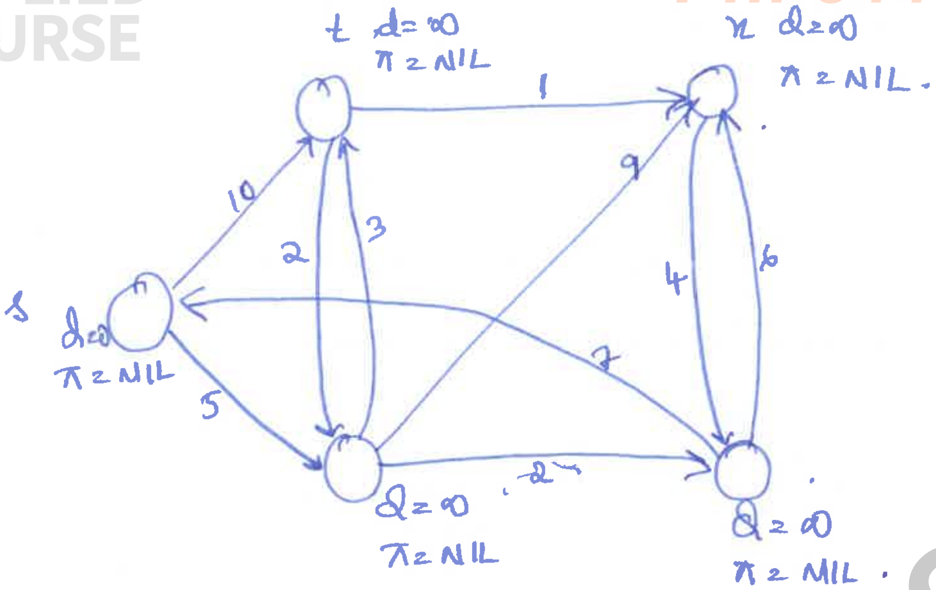
→ Line 3 the Min heap is constructed (which is a priority queue as well)

→ lines 4 to 8. the while loop is executed until the priority queue is empty.

- At each iteration the minimum element is extracted from the queue and added to the set S.

- All its neighbours are explored if a lighter/cheaper cost path is found for any neighbouring vertex then it is updated and its predessor information is updated as well.

Let us run the DIJKSTRA's on the sample graph given, the initialization is done




① - Initially s is removed from the Queue.

t. d is updated

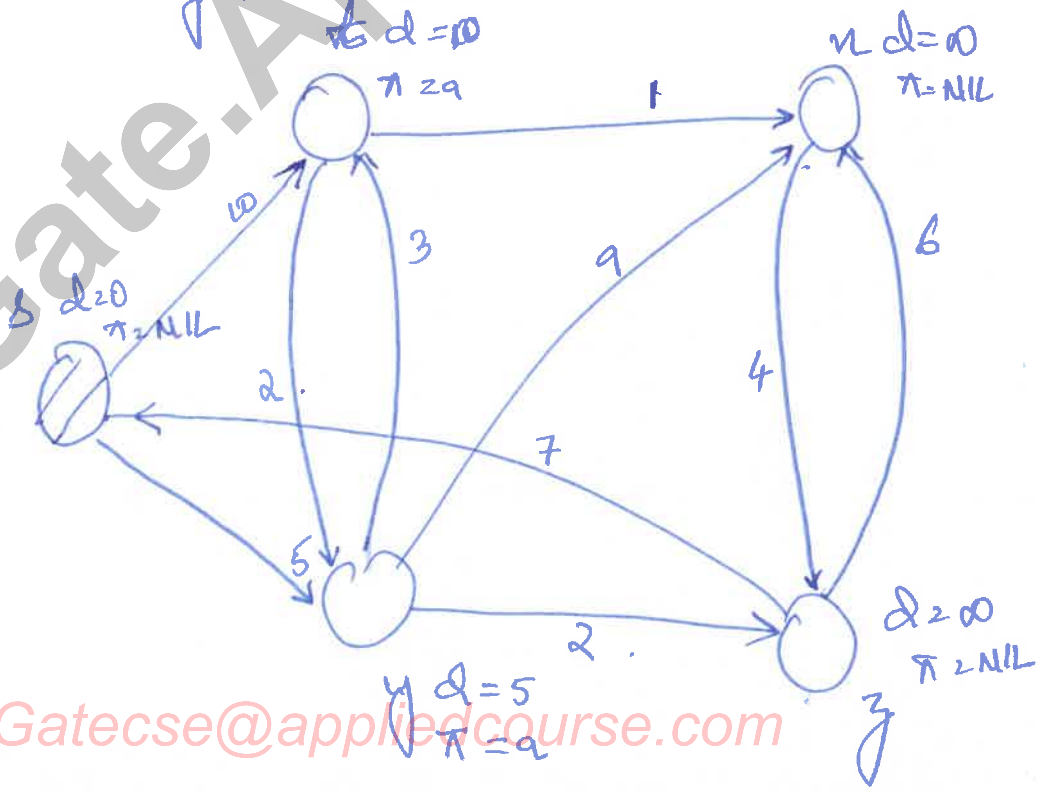
$t.d = 10$

$t.\pi = s$.

y is updated as well.

$y.d = 5$

$y.\pi = a$



(2) Now the node y is removed from the priority Queue.

(a) $t.d = 5+3 = 8$.

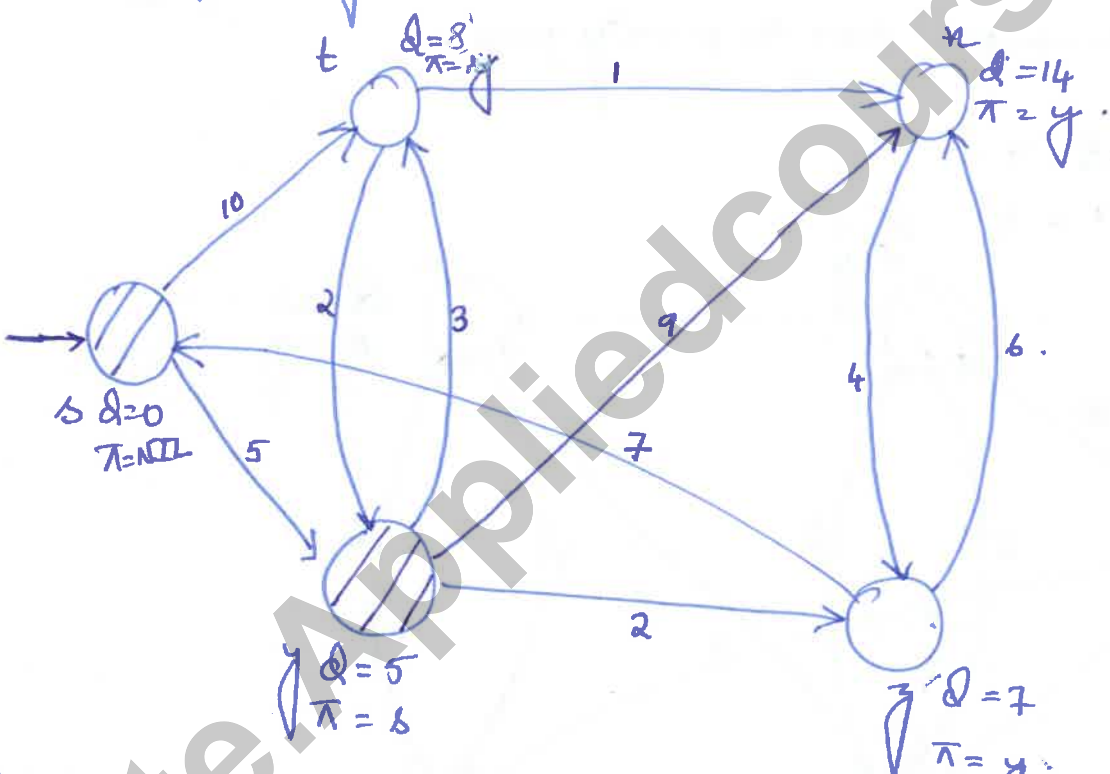
$t.\pi = y$.

(b) $x.d = 5+9 = 14$

$x.\pi = y$.

(c) $z.d = 5+2 = 7$

$z.\pi = y$.

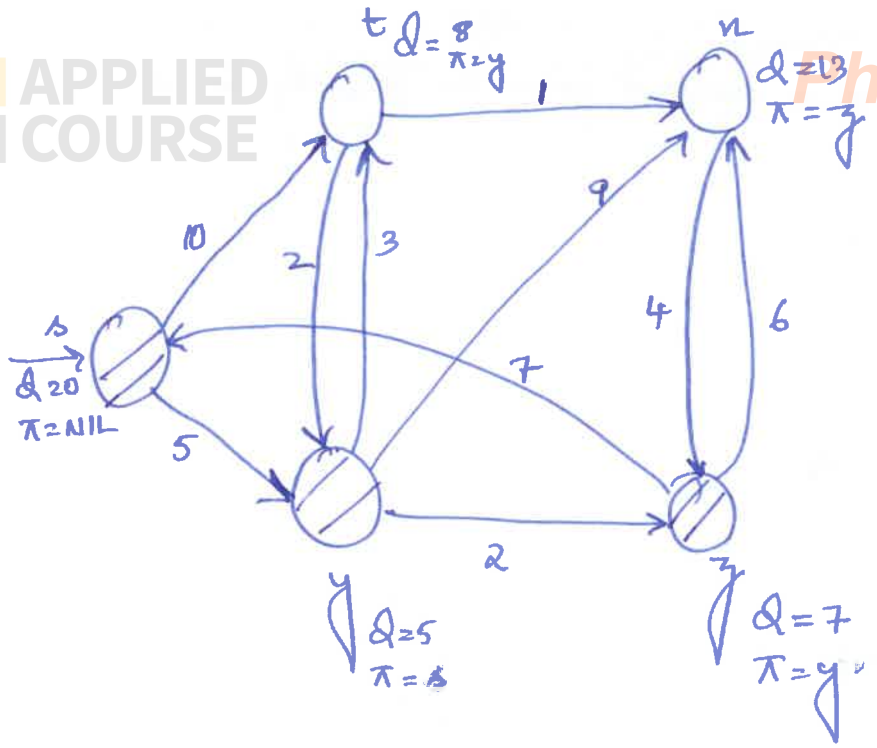



(3) Now z is removed from the priority queue.

(a) $x.d = 7+6 = 13$

$x.\pi = z$

$S = \{s, y\}$

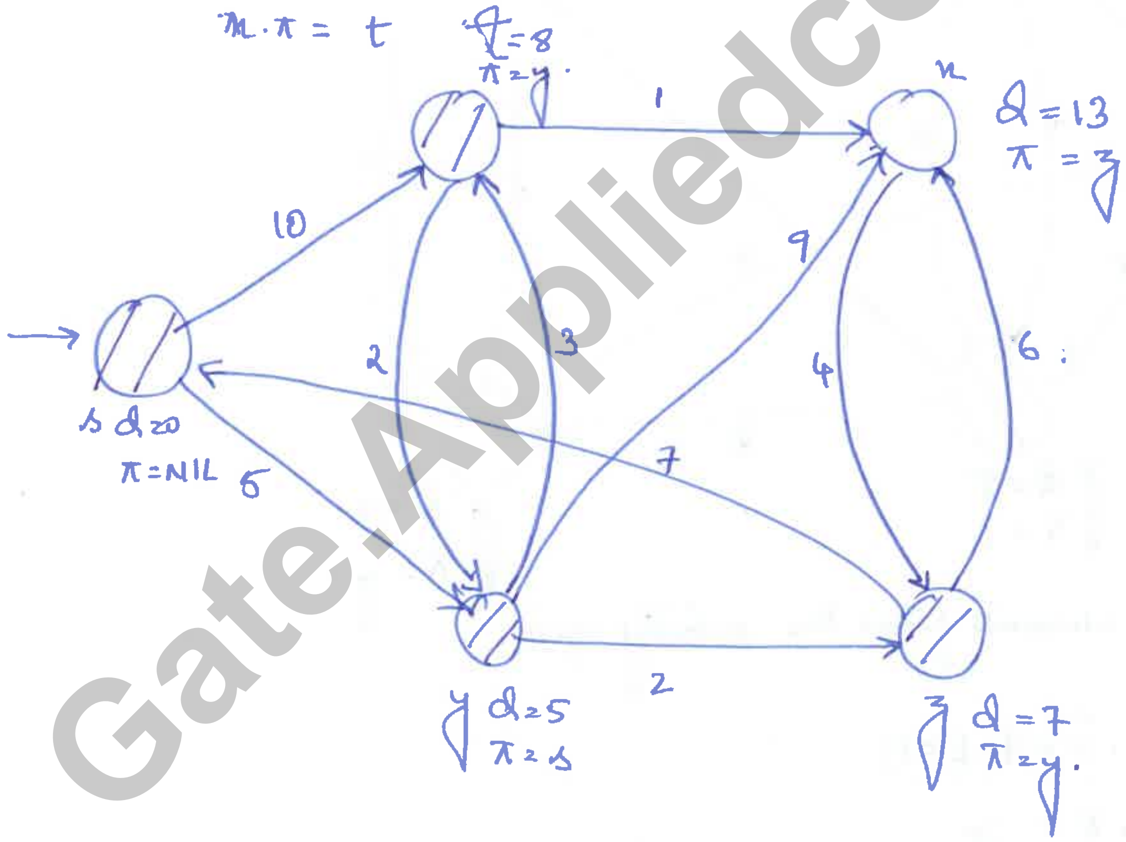



S = {s, y, z}

(4) Now t is removed from the priority queue

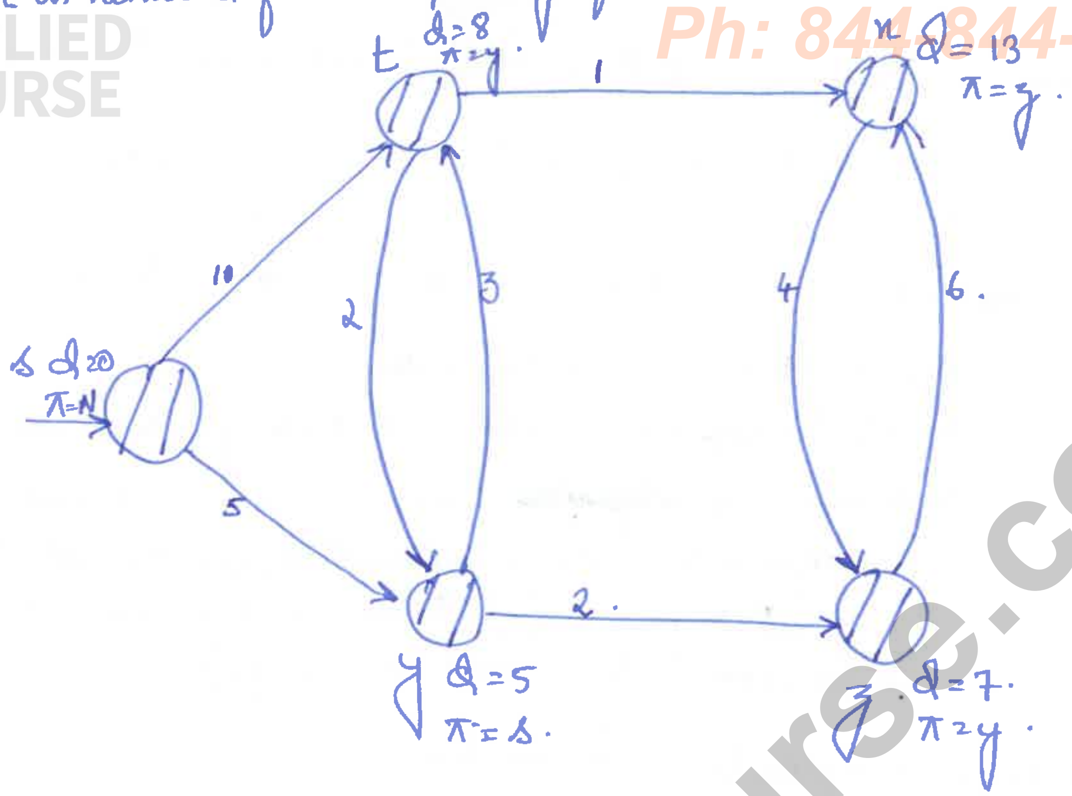
x.d = 9.

x.π = t




S = {s, y, z, t}

5) Now x is removed from the priority queue.




$S = \{s, y, z, t, x\}$

Now we can trace the shortest path to any vertex from that node back up to the root/source.

for example for vertex $z - (z \to y \to s)$

Time Complexity

- The initialize single source function goes through all the vertices of the graph it takes $O(V)$ time to execute

- Line 2. is of constant time.

- Line 3 is the heap construction which will take $O(n)$ time, for V vertices it will take $O(V)$ time.

- Line 5 is inside the loop extract min takes $O(\log(V))$ time it is executed for each vertex so it takes $O(V \log V)$ time.

- line 6 takes constant time as it is set union, it is executed for every vertex, therefore the time complexity $O(V)$

- The loop at lines 7-8 is executed a total of $E$ times because it is executed for each of the edges.
  - Inside the loop we are calling the Relax function which will execute for $\log(V)$ time as we are just updating the weight and predessor information, but the statement is executed for $O(E)$ times as it is within the loop, so time complexity would be $O(E \log V)$

Total time complexity is given by

$$= O(V) + O(V \log V) + O(V) + O(E \log V)$$

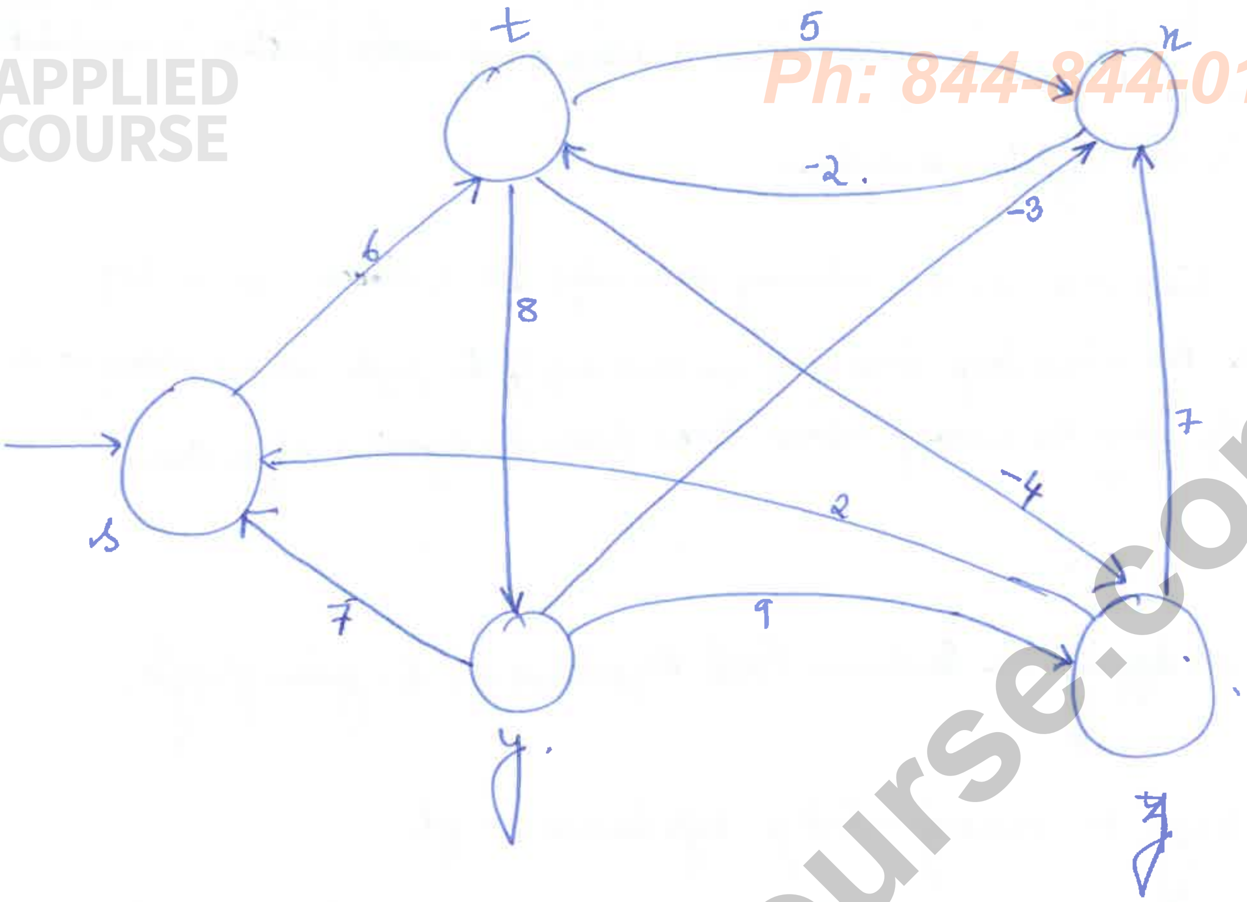
$$= \underline{\underline{O(E \log V)}}$$

---

## 62.3 BELLMAN FORD ALGORITHM

** - The Bellman ford algorithm works even if the edge weights are negative unlike the Dijkstra's Algorithm.

→ Also for the same single source shortest path problem.




BELLMAN-FORD$(G, w, s)$

```
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

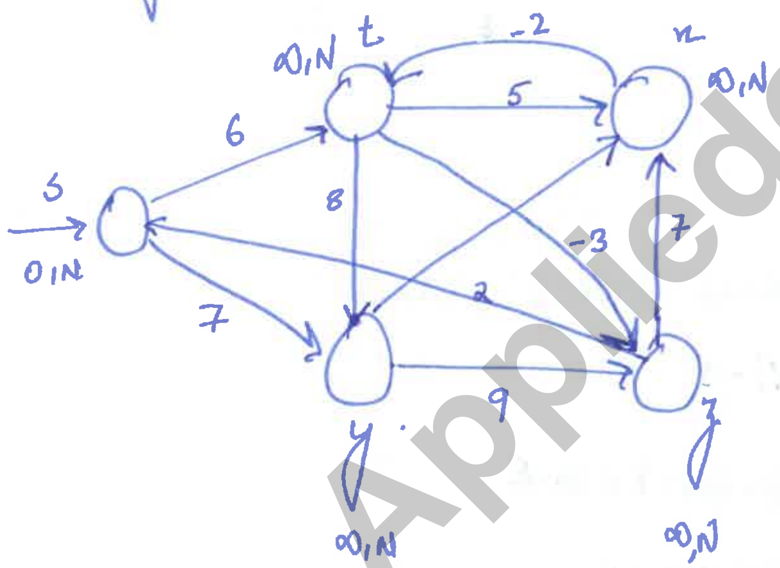
→ In line 1 we make of the same initialize single source function as mentioned in the Dijkstra's Algorithm.

→ In lines 2-4 we are relaxing each edge (|V|-1) times in the loop.

→ In the inner loop lines 3-4 for each edge of the graph we are relaxing each edge using the same function RELAX from the Dijkstra's algorithm.

Let us apply the Bellman Ford Algorithm on the given graph.

- Initially on running initialize single source we get.

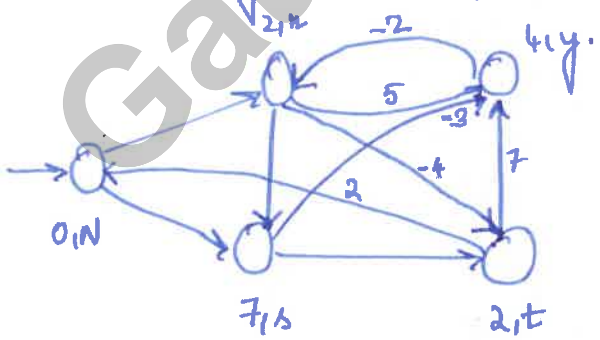



We have $|V| = 5$

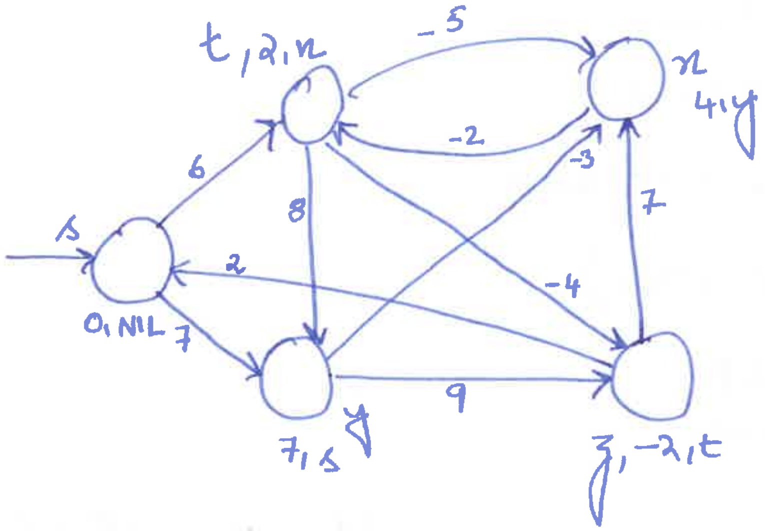
$(|V|-1) = 4$

We need to

① On relaxing all the edges once we get the following updated graph

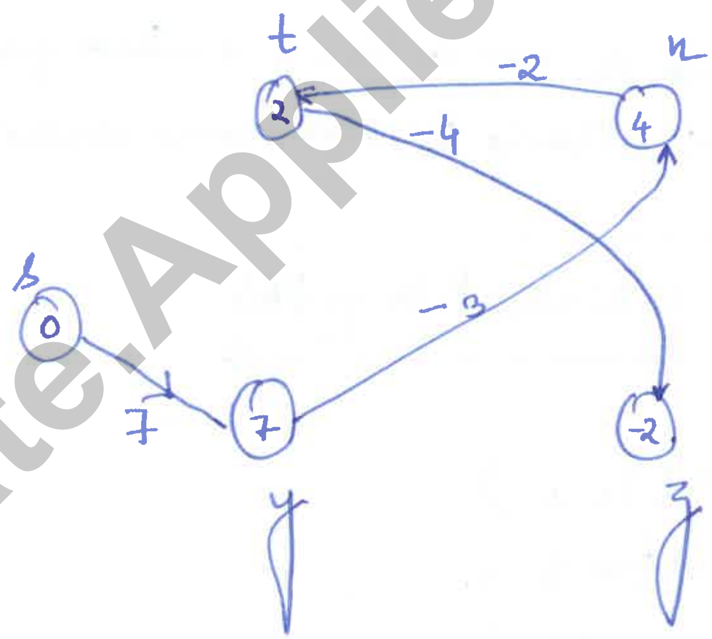


② → On relaxing all the edges for the second time we get the following graph with the updated weights.




③④ → On relaxing all the edges for the third and the fourth time we will get the same graph as above.

→ Below is the subgraph which shows the shortest path



Time complexity:-

- The outer loop runs $O(V)$ times
- The inner loop runs $O(E)$ times.
- The loop at lines 5-7 runs $O(E)$ times

Total time complexity $O(VE) + O(E)$

$= O(V \cdot E)$.

→ The loop in lines 5-7 check for -ve weight cycles, after (V-1) relaxations if there is still a chance of reducing the weight that means that there is a cycle of -ve weight.

→ A single source shortest path cannot contain a negative weight cycle.

→ In case a -ve cycle exist the loop in lines 5-7 returns false, in such a case we cannot determine the single source shortest path.

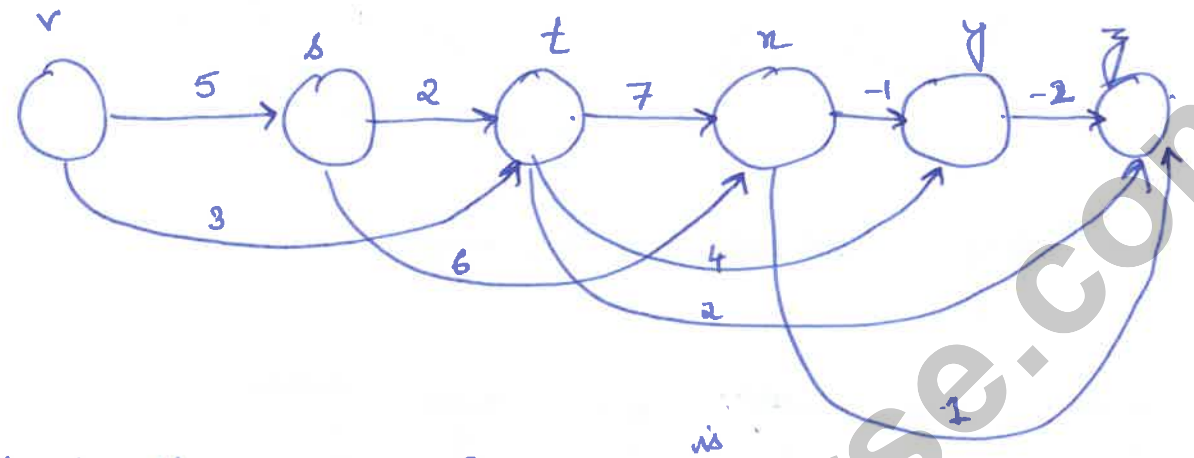
---

## 6.2.4 Shortest Path for DAG

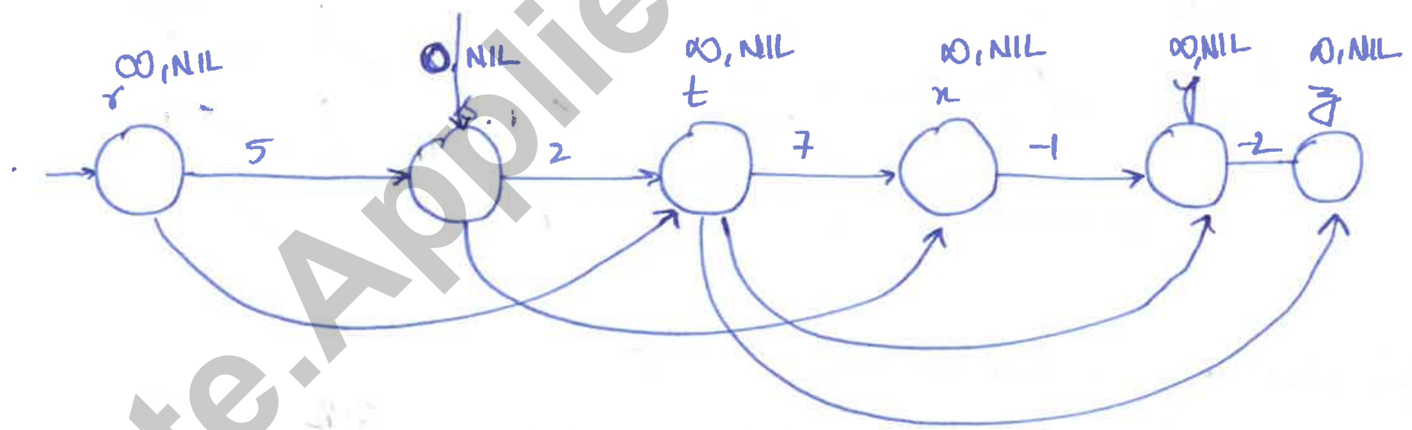
DAG-SHORTEST-PATHS (G, w, s)

1. Topologically sort the vertices of G.
2. INITIALIZE-SINGLE-SOURCE (G, s)
3. for each vertex u, taken in topologically sorted order
4. for each vertex $v \in G.Adj[u]$
5. RELAX (u, v, w)

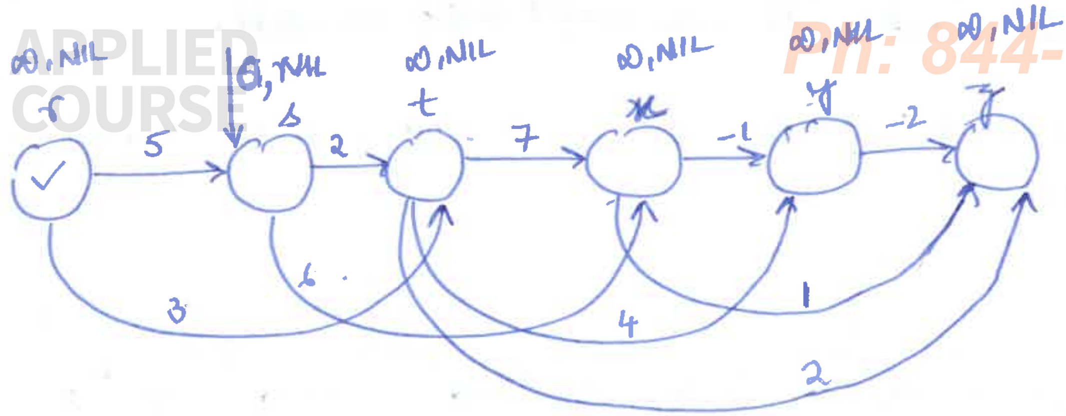
- The algorithm works with -ve weight edges as well.
- We know that topological sort can be run on any DAG in $O(V+E)$ by using DFS.



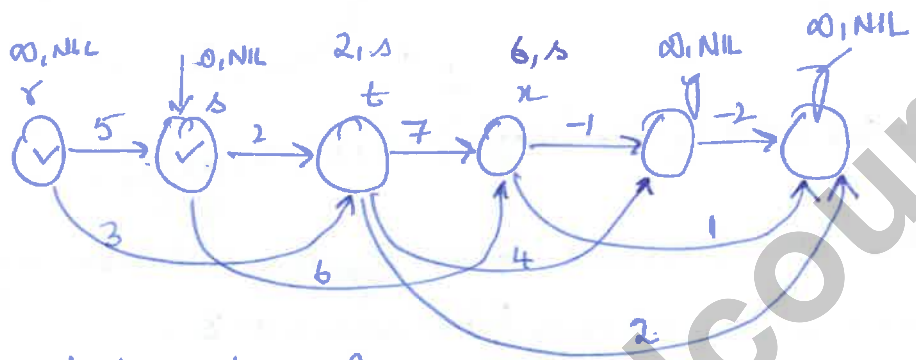

- The topologically sorted order of the Graph is $\{v, s, t, x, y, z\}$.
- Now let us run the DAG-SHORTEST-PATHS Algorithm on the above graph. after initialization we have. Here we have taken source as s.



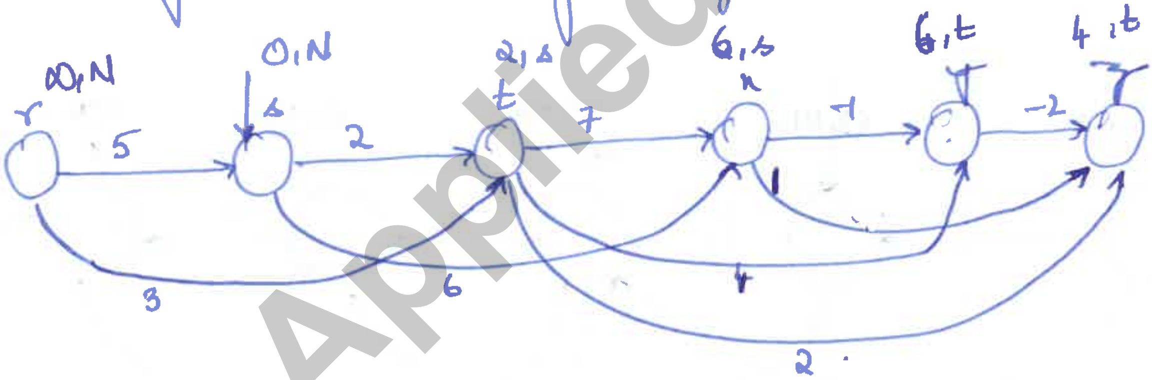

① starting with vertex r (in topological sorted order), we get the following graph.



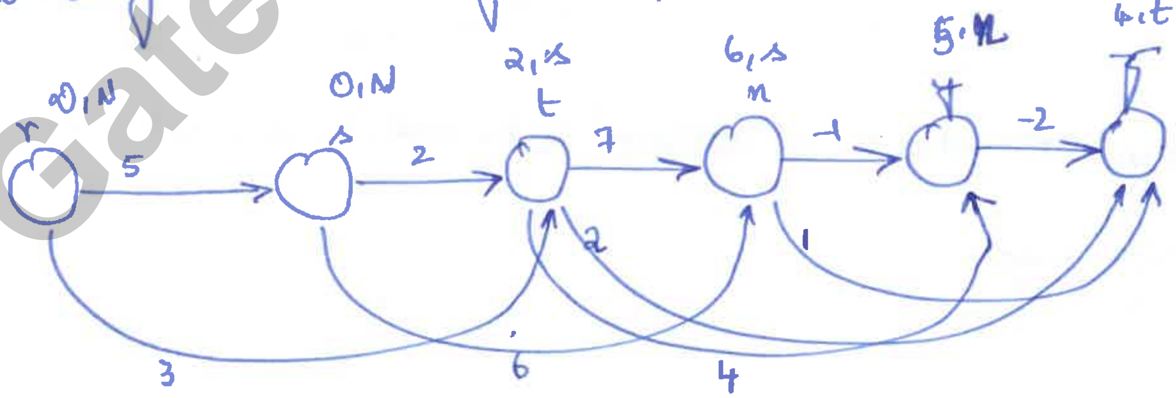

(2) Now taking s and relaxing the edges



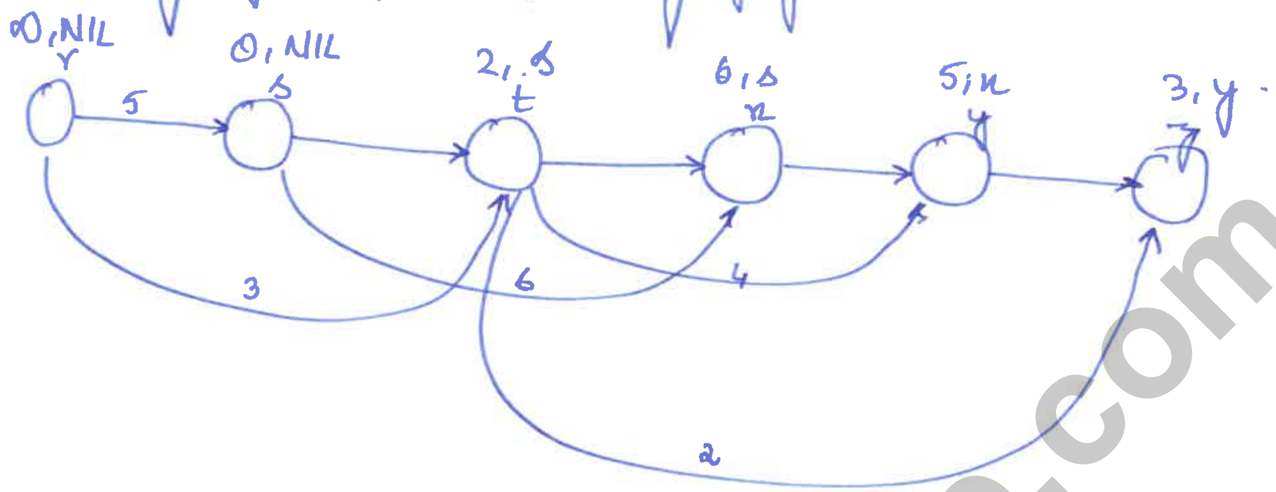

(3) Now taking t and relaxing the edges.




(4) Now taking x and relaxing the edges

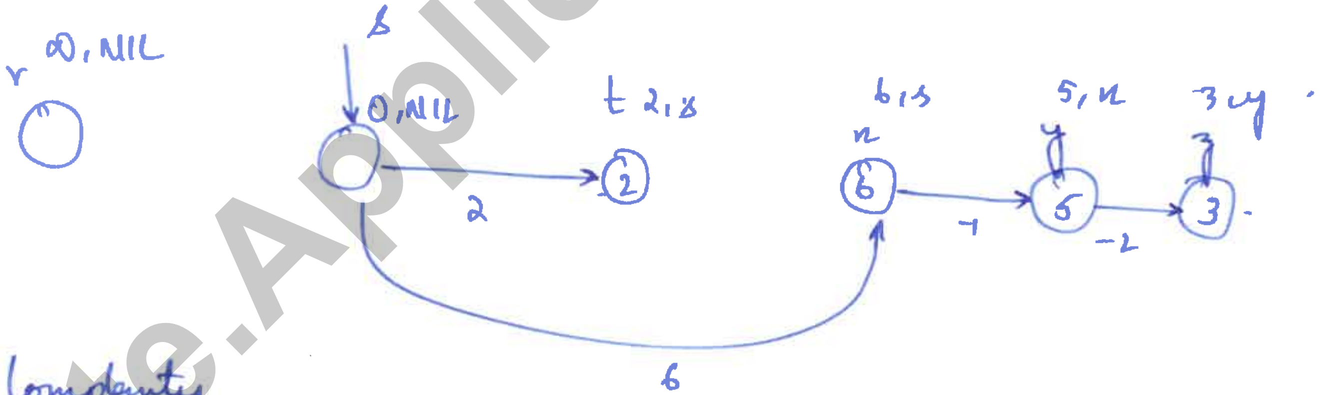


(5) Now relaxing y we get the following graph.




(6) Now z does not have any outgoing edges so it remains as the above graph only.

Following is the graph with the single source shortest paths.




Time Complexity

1. line 1 will be done by applying DFS $O(V+E)$
2. line 2 will take $O(V)$ time
3. lines 3-5 is a loop will execute $O(V)$ times
4. the inner loop will will execute $O(E)$ times and the total times relax will be called is $O(E)$ times.

Total time complexity $O(V+E)$

If we apply Bellman ford algorithm we can have $O(V \times E)$ time.
but because of the directed acyclic graph.

## 62.5 All Pairs Shortest Paths

### Matrix Operations

- Here we want to find the shortest path between every two possible vertices of the graph.

- The Bellman ford algorithm is a single source shortest path. If we run it for all the V vertices of the graph it will take $= V \times O(E \times V)$
$= O(EV^2)$ time

In case of a dense graph $E \simeq V^2 = O(V^4)$ time
and in case of a sparse graph it is $O(EV^2)$ time.

Can we do better??

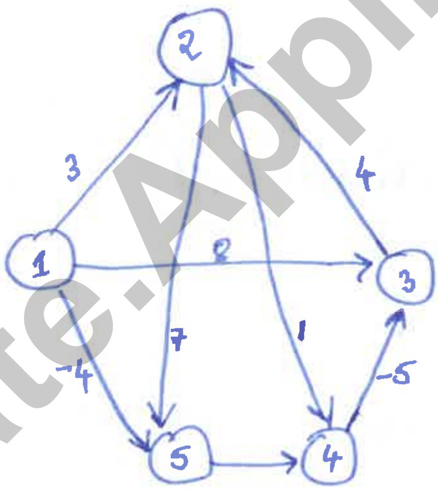
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

1. $n = W.rows$
2. $L^{(1)} = W$
3. for $m = 2$ to $n-1$
4. let $L^{(m)}$ be a new $n \times n$ matrix
5. $L^{(m)} = $ EXTENDED-SHORTEST-PATHS$(L^{(m-1)}, W)$
6. return $L^{(n-1)}$.

# EXTENDED-SHORTEST-PATHS(L, W)

```
n = L.rows
let L' = (l'_ij) be a new n x n matrix.
for i = 1 to n
    for j = 1 to n
        l'_ij = ∞
        for k = 1 to n
            l'_ij = min(l'_ij, l_ik + w_kj)
return L'
```






$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & \infty & -5 & 0 & \infty \\
5 & \infty & \infty & \infty & 6 & 0
\end{array}
$$

The weight matrix is constructed by

$$
w_{u,v} = \begin{cases} w(u,v) & \text{if edge}(u,v) \text{ exists} \\ 0 & \text{if it is } u = v \\ \infty & \text{if no edge}(u,v) \text{ exists} \end{cases}
$$

On running the Slow All Pairs Shortest Paths we get the following matrix

$$L^1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

For the second iteration we have the following matrix after the iteration

$$L^2 = \begin{bmatrix} 0 & 3 & 8 & 2 & -1 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{bmatrix}$$

The third iteration we have to use $L^{m-1}$ and $W$ some of the entries are.

$$L^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \_ & \_ & -3 & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & -2 \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix} \end{array}$$

We have to repeat this for $(n-1)$ $(5-1) = 4$ iterations in a similar way.

<u>Time complexity</u>.

- The time complexity of the extended shortest paths is $O(N^3)$ because we have 3 nested loops.

— In the Slow All pairs shortest paths

The loop lines 3-5 executes $O(V)$ times

— Line 4 will require $O(V^2)$ time

: Line 5 is $O(V^3)$

$\therefore$ Total time complexity $= \underline{\underline{O(V^4)}}$

$O(V^4)$ is the same time which Bellman Ford algorithm required when run on each of the vertices on a dense graph.

— There is a faster variant of the all pairs shortest path.

In the Slow APSP (All Pairs Shortest Paths)

First Step we calculate $L_1^{(1)} = W$

Second step $L^{(2)} = ESP(L^{(1)}, W) - O(V^3)$

$L^{(3)} = ESP(L^{(2)}, W) - O(V^3)$

$\vdots$

$L^{(n-1)} = ESP(L^{n-2}, W)$

$O(V^4)$

If we use the faster version it makes use of a property of ESP

$L^{2n} = ESP(L^n, L^n)$

In the first step $L^1 = W$

In the second step $L^2 = ESP(L^1, L^1)$

In the third step $L^4 = ESP(L^2, L^2)$

$\vdots$

$L^8 = ESP(L^4, L^4)$

$L^{16} = ESP(L^8, L^8)$

$\vdots$

It will take $\lceil \log(n-1) \rceil = O(\log V)$ steps to reach $\underline{\underline{n-1}}$

The time complexity reduces to $O(V^3 \log V)$

# 62.6 FLOYD WARSHALL ALGORITHM



- All pairs shortest paths problem.
- It achieves $O(V^3)$ time complexity
- It is a Dynamic Programming algorithm.

## FLOYD-WARSHALL(W)

1. $n = W.rows$.
2. $D^{(0)} = W$.
3. for $k = 1$ to $n$
4. let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5. for $i = 1$ to $n$
6. for $j = 1$ to $n$
7. $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$.
8. return $D^{(n)}$.



- At each step $k$ we have two matrices $D^k$ and $\pi^k$, the distance and predecessor matrix respectively. (The predecessor matrix is not written but it has to be updated).

- For $d_{ij}^k$ we use the expression $\min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

example $d_{42}^1 = \min(d_{42}^{1-1=0}, d_{41}^0 + d_{12}^0) = \min(\infty, 2+3) = 5$ $(\pi_{42} = 1$ also$)$

(1) On the initial we have.

$$D^{(0)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad \pi^{(0)} = \begin{bmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

(2) After the first iteration we have.

$$D^{(1)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad \pi^{(1)} = \begin{bmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

(3) After the second iteration we have.

$$D^{(2)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad \pi^{(2)} = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

(4) After the third Iteration we have.

$$D^{(3)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^{(3)} = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

(5) After the fourth Iteration we have.

$$D^{(4)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad \pi^{(4)} = \begin{bmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{bmatrix}$$

(6) After the fifth Iteration we have.

$$D^{(5)} = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad \pi^{(5)} = \begin{bmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{bmatrix}$$

- To know the actual shortest path between any two vertices can be traced using the predecessor matrix $\pi$.

for example for shortest path in between (3,4)

$\pi^{(5)}_{3,4} = 2$. Predecessor is 2

$\pi^{(5)}_{3,2} = 3$ Predecessor is 3.

$$(3) \xrightarrow{4} (2) \xrightarrow{1} (4)$$

cost is = 5

also $D^{5}_{34} = 5$.

for (5,2).

$\pi^{(5)}_{(5,2)} = 3$ $\quad \pi^{(5)}_{(5,3)} = 4$ $\quad \pi^{(5)}_{(5,4)} = 5$

$$(5) \rightarrow (4) \rightarrow (3) \rightarrow (2).$$

## Time Complexity :

- There are 3 nested loops each of which executes $O(V)$ times and inside the inner most loop we are doing constant time operations

$\therefore$ Total time complexity = $\underline{O(V^3)}$

# 62.7 Solved Problem



## GATE 2013

Q) What is the time complexity of Bellman Ford single source shortest path algorithm on a complete graph of n vertices?

(A) $\theta(n^2)$ (B) $\theta(n^2 \log n)$ (C) $\theta(n^3)$ (D) $\theta(n^3 \log n)$.

Soln We know time complexity of Bellman Ford algorithm $O(EV)$

but we do not have a similar option for a dense graph.

$E \simeq V^2$ it goes to $O(V^3) = O(n^3)$ option C.

---

## 62.8 Solved Problem GATE 2006.

Q) To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time the data structure to be used is

(A) Queue
(B) Stack
(C) Heap.
(D) B-Tree.

Soln :- Dijkstra's algorithm takes $O(E \log V)$ time which is not linear in E or V

But if we apply BFS on an unweighted graph we will get the single source shortest path.

Intuitively.

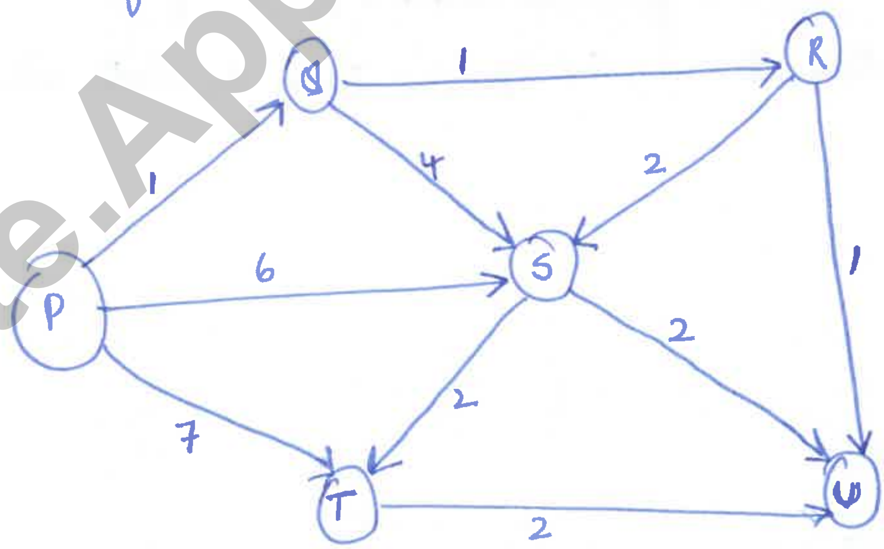
Dijkstra's algorithm reduces to BFS for the unweighted graphs. and the Time Complexity O(E+V) which is linear in V and E.

In BFS we make use of a Queue. option A is correct.

---

## 62.9 Solved Problem GATE 2008

Q) Suppose we run Dijkstra's single source shortest path algo on the following edge weighted directed graph with vertex P as the source. In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized.



A. P, Q, R, S, T, U.

B. P, Q, R, U, S, T

C. P, Q, R, U, T, S

D. P, Q, T, R, U, S

Initially

$S = \phi$

① P is removed

$S = \{P\}$. $Q.d = 1$
$T.d = 7$.

② Q is removed

$S = \{P, Q\}$. $R.d = 2$.
$S.d = 5$

③ R is removed. (option D can be eliminated)

$S = \{P, Q, R\}$

$U.d = 3$

$S.d = 4$

④ U is removed (option A can be eliminated)

$S = \{P, Q, R, U\}$

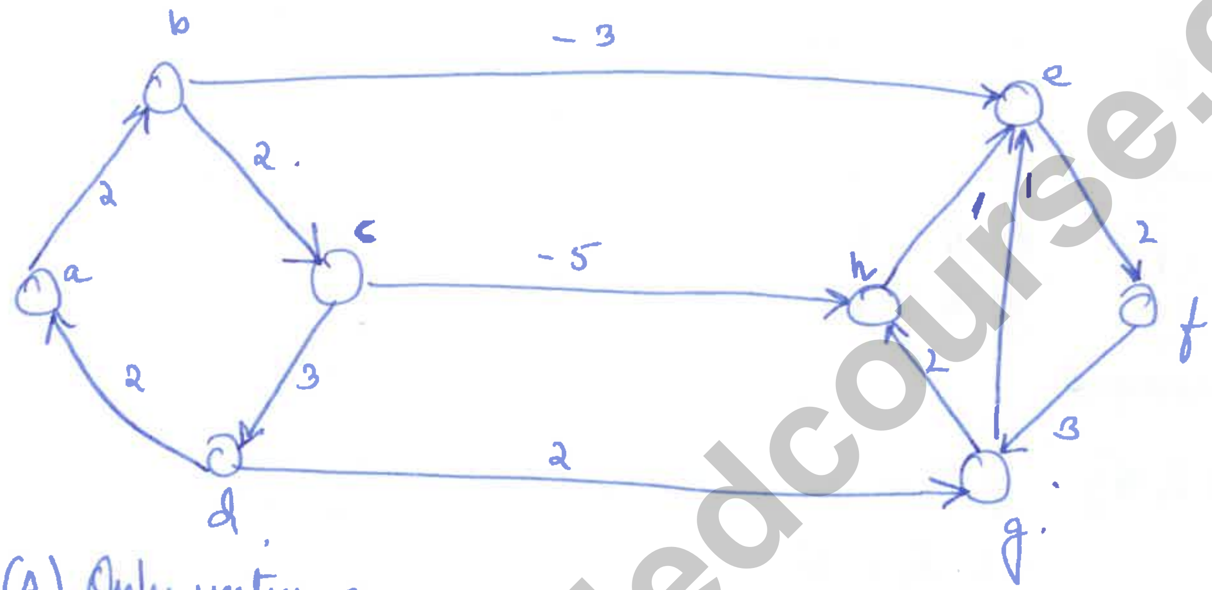
⑤ S is is removed (option C can eliminated)

B is the correct Answer.

# 62.10 Solved Problem GATE 2008



Q) Dijkstra's single source shortest path algorithm when run for the following graph from vertex a, it computes the correct shortest path distance to:




(A) Only vertex a

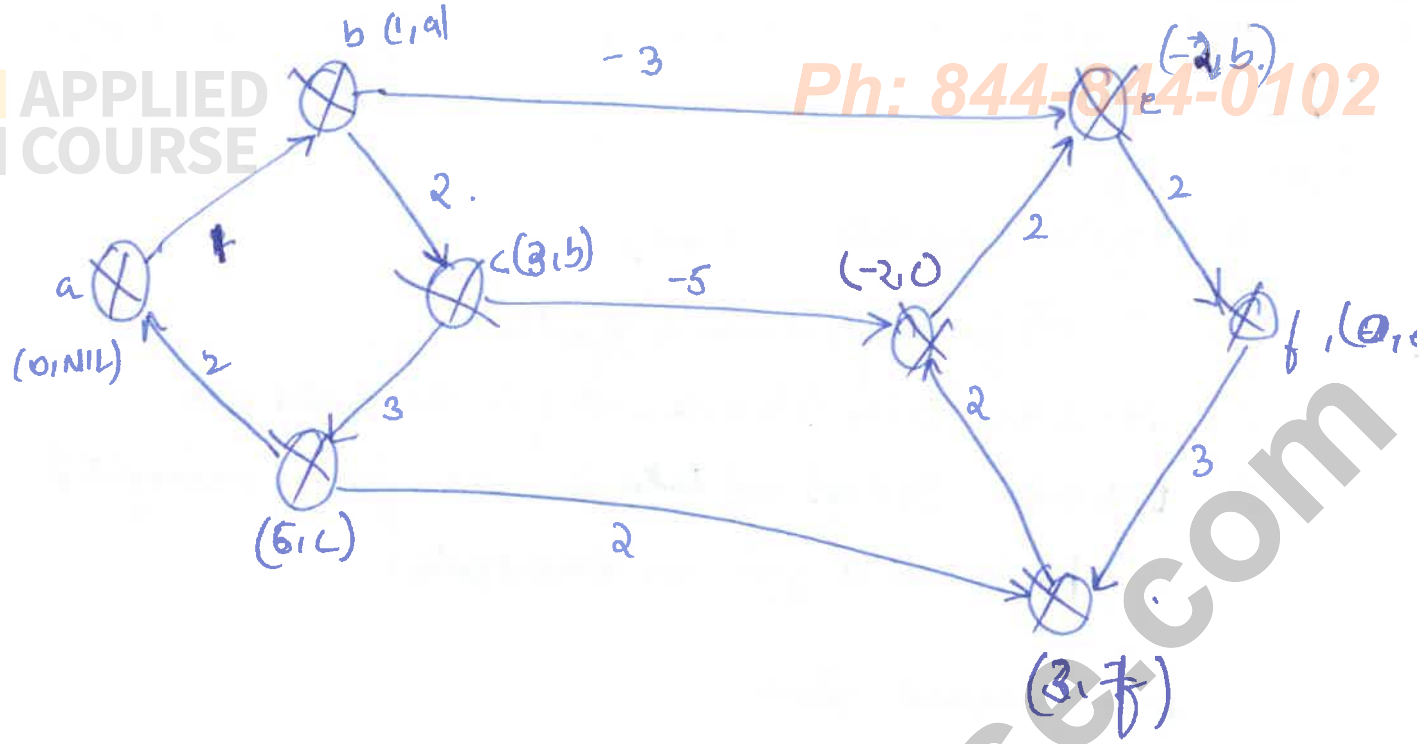
(B) only for vertices a, e, f, g, h.

(c) only vertices a, b, c, d,

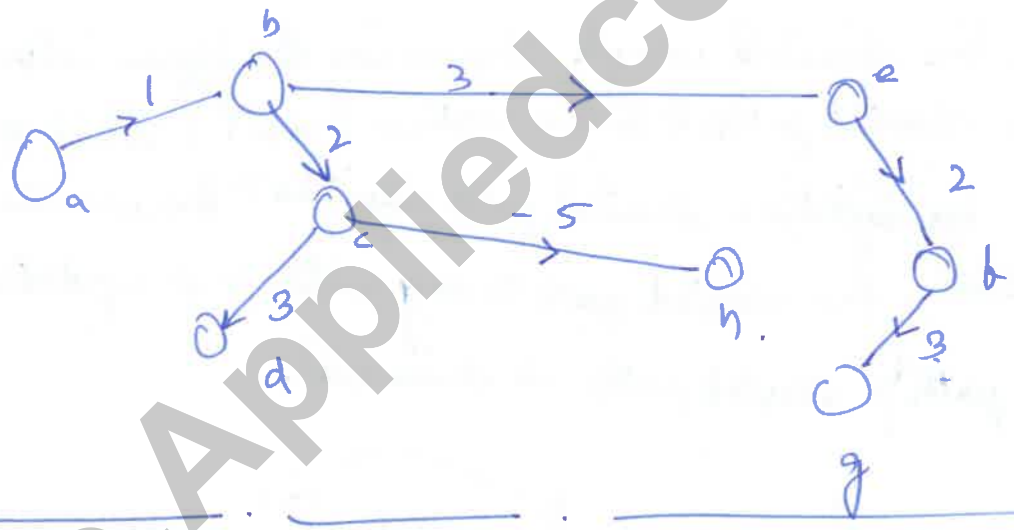
(D) All the vertices.

The Dijkstra's algorithm is guaranteed to work only on all +ve edges graph, if there is a -ve edge it may or may not work.

On running the Dijkstra's we get the following




- For the above graph we are able to find the shortest paths for all the pairs.




## 62.11 Solved Problem GATE 2007

→ In an unweighted undirected connected graph, the shortest path from a node S to every other node is computed most efficiently by

(A) Dijkstra's Algorithm starting from S

(B) Warshall's Algorithm

(C) Performing DFS starting from S.

(D) Performing BFS starting from S.

Ans :-

A - Dijkstra's Algo takes $O(E \log V)$

B. $O(V^3)$ for Floyd Warshall's Algorithm

C. DFS takes $O(E+V)$ but does not give the shortest path

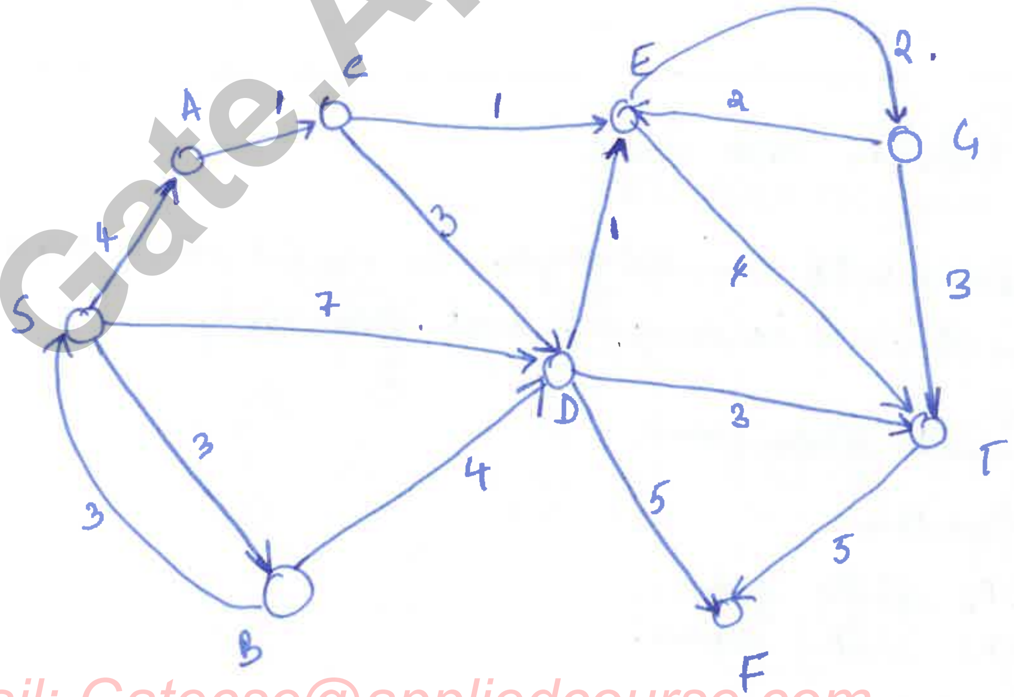
D. BFS takes $O(E+V)$ and it works for an unweighted graph as well to give the shortest path.

D is the correct option.

---

## 62.12 Solved Problem GATE 2012

Q) Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijkstra's shortest path algorithm? Assume that at every iteration, the shortest path to a vertex v is updated only when a strictly shorter path is discovered.



A. SDT

B. SBDT

C. SACDT

D. SACET

Solution

S.d = 0. S.π = NIL.

(1) S is removed from the Q.

A.d = 4. A.π = S

B.d = 3. B.π = S.

D.d = 7 C.π = S

(2) B is removed.

(3) A is removed.

C.d = 5 C.π = A

(4). C is removed.

E.d = 6. E.π = C

(5) E is removed.

T.d = 10 T.π = E

G.d = 8 G.π = E

(6) D is removed.

F.D = 12

F.π = D.

T is not updated as it is not less than 10.

(7) G is removed.

As the question

(8) T is removed.

As the question is about T we can stop here.

T.π = E

E.π = C

C.π = A

A.π = S.

∴ The path is SACET

option D.

# 62.13 Solved Problem GATE 2009



Q) Which of the following statements is/are correct regarding Bellman Ford shortest path algorithm?

P. Always finds a negative weight cycle if one exists.

Q. Finds whether any negative weighted cycle is reachable from the source.

(A) P only.

(B) Q only.

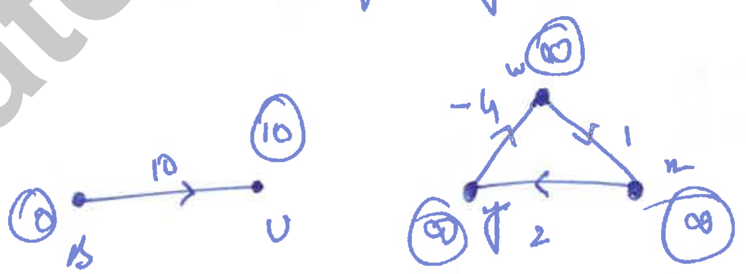
(C) Both P and Q.

(D) Neither P nor Q.

Ans:

We cannot ensure that Bellman Ford Algorithm will always detect the -ve edge cycle.

Let us consider the following graph.




On Running from s we are still not able to detect -ve weight cycle w x y because it is not reachable, therefore only reachable ones can be detected. option B is correct

Q) Let G(V, E) an undirected graph with positive weight edges. Dijkstra's single-source shortest path algorithm can be implemented using the binary heap data structure with time complexity?

A. $O(|V|^2)$

B. $O(|E| + |V| \log |V|)$

C. $O(|V| \log |V|)$

D. $O((|E| + |V|) \log |V|)$.

Ans :-

Actually the time complexity is $O(E \log V)$ but it is not present in the options so actually the time complexity is

$$O(\underbrace{V \log V}_{\text{for delete}} + \underbrace{E \log V}_{\text{for update option}})$$

Most closest option which is required is option D $\underline{O((|E|+|V|) \log |V|)}$.

# 62.15 Solved Problem Gate 2016-1



Q) Let G be a weighted connected graph (undirected) with distinct positive edge weights. If every edge weight is increased by the same value, then which of the following statements is/are TRUE?

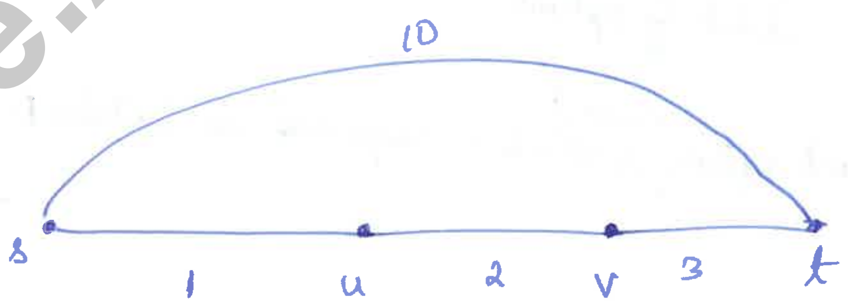
P: Minimum Spanning tree of G does not change.

Q: Shortest path between any pair of vertices does not change.

**Answer:**

Let us check for statement Q.

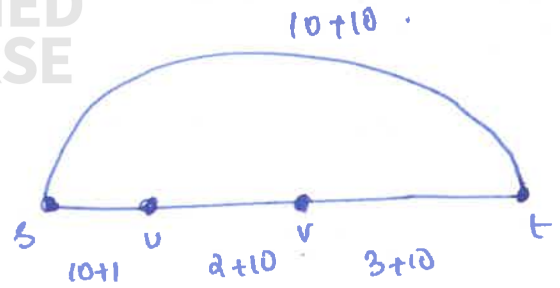
lets take the following sample graph




shortest path from s to t = 1+2+3 = 6. s−u−v−t

if we add 10 on each edge:

We get



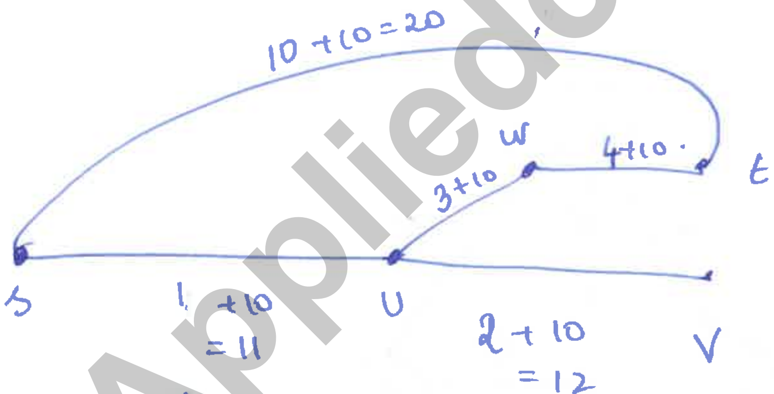

The shortest path now is $s-t = 20$

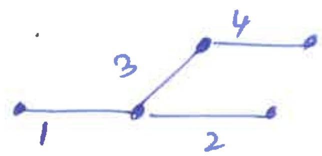
Because the length is only one edge.

Therefore Q is false.

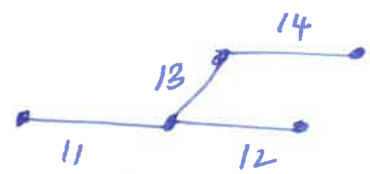
Now checking for statement P.

Lets consider the following examples.




Now if we add 10 to each edge




– The MST remains unchanged. If we try different options we do not get any other options but we cannot get any other graphs in which the MST before and after the increase are different.

This is because every MST of connected graph will have $|V| - 1$ edges which remains unchanged. ∴ P is true. option A is correct.

# 62.16 Solved Problem GATE 2016

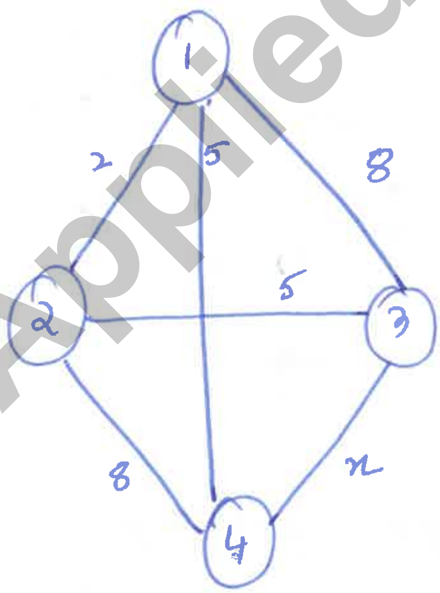


Q) Consider the weighted undirected graph with 4 vertices where weight of edge $\{i, j\}$ g is given by the entry $W_{ij}$ in the matrix

$$W = \begin{bmatrix} 0 & 2 & 8 & 5 \\ 2 & 0 & 5 & 8 \\ 8 & 5 & 0 & x \\ 5 & 8 & x & 0 \end{bmatrix}$$

The largest possible value of $x$ for which the shortest path in between some pair of vertices will contain the edge with weight $x$ is ______.

Ans






$1 \rightsquigarrow 2 = 2$

$1 \rightsquigarrow 3 = 8$ or $5 + x$ ← to be shortest $x$ should be $x < 3$ — $x = 2$

$1 \rightsquigarrow 4 = 5$

$2 \rightsquigarrow 3 = 5$

$2 \rightsquigarrow 4 = 8$ or $5 + x$ also here $x < 3$ — $x = 2$

$3 \rightsquigarrow 4 : x$ or $5 + 8$ ← $\underline{x < 13}$ — $x = 12$ — The largest possible value of $x = 12$

– Algorithm design strategy for designing more efficient algorithms.

PROBLEM :- Fibonacci Numbers.

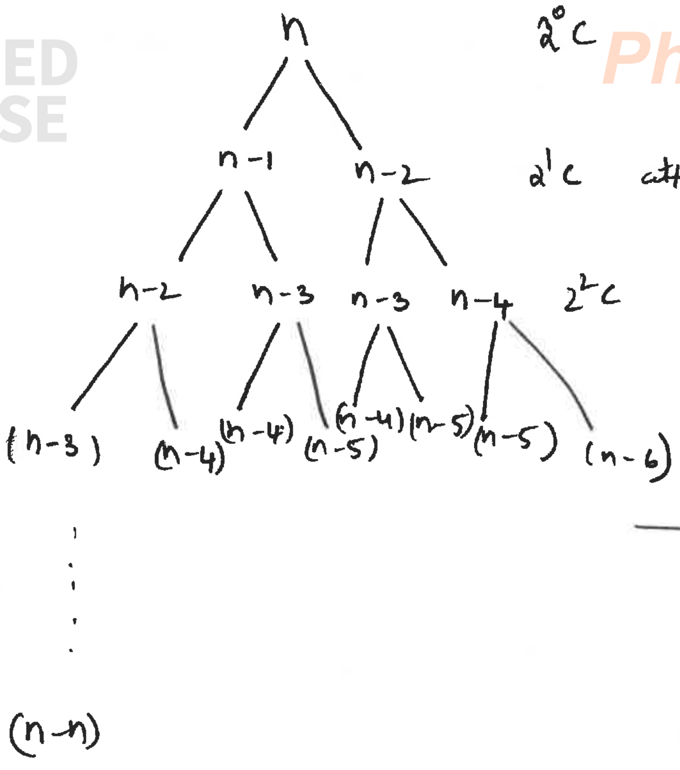
$$Fib(n) = \begin{cases} n \text{ if } & n = 0 \text{ or } 1 \\ Fib(n-1) + Fib(n-2) & \text{if } n >= 2. \end{cases}$$

Recursive definition

```
int Fib(n)
{
    if (n <= 1)
        return n;
    else
        return Fib(n-1) + Fib(n-2);
}
```

Recurrence Relation

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{if } n >= 2 \\ 1 & \text{if } n <= 1 \end{cases}$$




$2^0 c$ If for one fn call c time is required

$2^1 c$ at first level of

$2^2 c$ at $2^{nd}$ level.

$2^3 c$

$2^{(n-1)} \cdot c$

$\{2^0 + 2^1 + \dots 2^{n-1}\} \cdot c$.

$\Rightarrow \{2^n - 1\} \cdot c$.

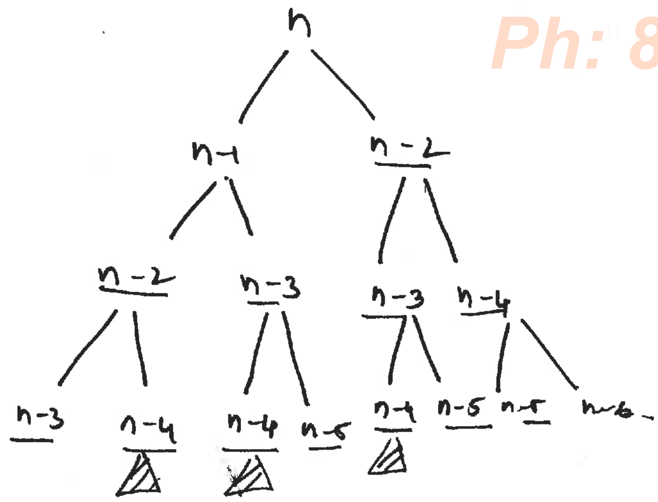
$\Rightarrow \underline{\underline{O(2^n)}}$. Exponential time complexity

– Because the maximum ht of the recursion tree = n, because at each level there is a decrease of 1.

Stack Space $O(n)$

→ Can we do better than this? in terms of time

Use Dynamic Substructure




Two properties at use in Dynamic Programming

① Optimal Substructure

$$f(n) = f(n-1) + f(n-2)$$

A problem is expressed interms of itself but in smaller subproblems

② Overlapping Subproblems :-

Sub problems occurs multiple times in the recursion tree.

Idea :- Compute the repeating fib(n-4) once and store and reuse it instead of recomputing it every time.

→ Two approaches for Dynamic Programming

① Top Down. {Memoization}

② Bottom Up. {Tabulation}

## ① Top down - Memoization

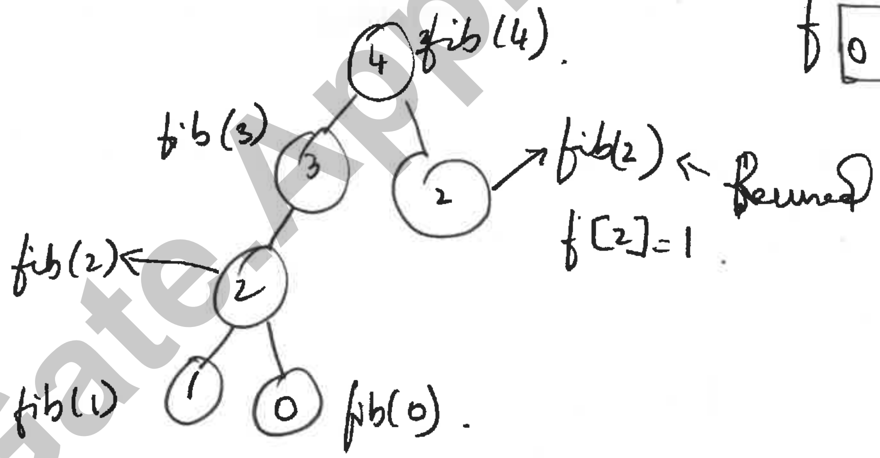
f[ ] ← array in which I will store fib[i], initialize the array to

$f[0] = 0$;

$f[1] = 1$;

$f[i] = \text{NIL}$ if $i > 1$.

```
int fib(n)
 {
    if f[n] == NIL
          f[n] = fib(n-1) + fib(n-2)
    return f[n]
 }
```




f

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 |

## (2) BOTTOM-UP [Tabulation]





f[ ] :- array

f[0] = 0 f[1] = 1 ; f[i] = NIL if i > 1

```
int fib(n)
{
    for(i = 2; i <= n; i++)
    {
        f[i] = f[i-1] + f[i-2]
    }
    return f[n];
}
```

Iteration

Top down (recursive)

n
n-1
n-2
⋮
2
1
0

Bottom up (iteration)

– For each i > 1 fib(i) is evaluated only once

∴ Total time complexity $O(n)$

Space complexity :- Stack + array

$O(n) + O(n)$

$= O(n)$.

2/14 Longest Common Subsequence (LCS) (33 mins)

$S_1$ ABCDGH.
$S_2$ AEDFHR.

LCS ($S_1, S_2$) = ADH of length 3 of maximal length.

AD, AH are also common subsequences but not the longest ones.

eg 2

$S_1$ = AGGTAB
$S_2$ = GXTXAYB

LCS(S1, S2) = GTAB of len 4

Applications of LCS.

1. Genomics :-

Genetic code of a ~~pof~~ person

Genomic code of a person { $S_1$ = AGTC ..... Billions of characters
$S_2$ = ATCGTCA .......

LCS is the similarity between the two.

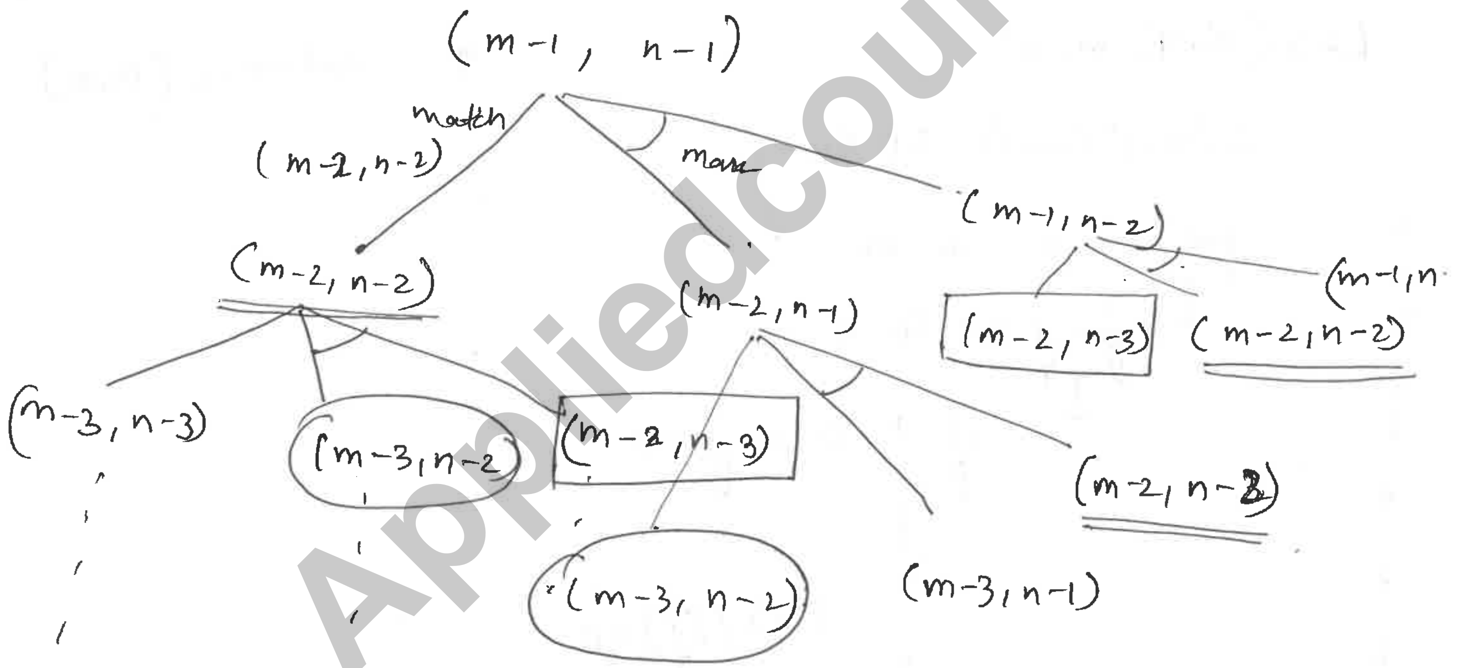
2) diff command : To calculate the file difference in UNIX systems.

S1 : X[0, 1, 2, ...... m-1] - len.
S2 : Y[0, 1, 2 ....... n-1]

## Recursion formula for LCS

$$LCS(m-1, n-1) = \begin{cases} LCS(m-2, n-2) + 1, & \text{if } X[m-1] == Y(n-1) \\ \max\left\{ \begin{array}{l} LCS(m-1, n-2), \\ LCS(m-2, n-1) \end{array} \right\} & \text{Otherwise} \end{cases}$$

## Recursion Tree




(2) We have overlapping sub problems here.

ht of the tree $[(m-1) + (n-1)]$

No of At each level.

here in the recursion tree if the last character is equal, then we evaluate $(m-2, n-2)$ or else we do $\max((m-2, n-1), (m-1, n-2))$

At each level for any of the problems we have 2 cases.

At level $l$ we have $2^l$ such cases or $3^l$ sub problems.

Which are exponential in number.

- Total no of sub problems would be $(3^{h+1} - 1) = O(3^h)$

By using Dynamic Programming this is reduced drastically.

<u>Bottom Up Dynamic Programming Approach.</u>

```
LCS(X, Y, m, n)
    L(m+1)(n+1) 2D Array.
    for i = 0 to m.
    {
        for j = 0 to n.
        {
            if i = 0 or j = 0
            {
                L[i][j] = 0
            }
            else if (X[i-1] == Y[j-1]).
            {
                L[i][j] = L[i-1][j-1]+1 // Last Character Match case.
            }
            else.
            {
                L[i][j] = max {L[i-1][j], L[i][j-1]}
            }
        }
    }
    return L[m, n]
```

Time Complexity.

- The outer loop is executed m times
- The inner for loop is executed n times
- Inside the inner loop 3 conditional statements are executed in constant time.

$\therefore$ Time Complexity $= O(m*n)$.

Space Complexity : Space required is for the tabulation array of Order $m \times n$.
$= O(mn)$

## 64.3 LCS Example

$S_1 = QPQRR.$

$S_2 = PQPRQRP$

$$LCS(m,n) = \begin{cases} 1 + LCS(m-1, n-1) & \text{if } S1[m] = S2[n] \\ \max(LCS(m, n-1), LCS(m-1, n)) & \text{Otherwise} \end{cases}$$

| $S_2 \downarrow$ / $S_1 \rightarrow$ | | 1 Q | 2 P | 3 Q | 4 R | 5 R |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0. |
| 1 P | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 Q | 0 | 1 | 1 | 2 | 2 | 2 |
| 3 P | 0 | 1 | 2 | 2 | 3 | 2 |
| 4 R | 0 | 1 | 2 | 2 | 3 | 3 |
| 5 Q | 0 | 1 | 2 | 3 | 4 | 3 |
| 6 R | 0 | 1 | 2 | 3 | 4 | 4 |
| 7 P | 0 | 1 | 2 | 3 | 4 | 4 |

LCS (1,1)

$S1[1] = Q$

$S2[1] = P$ Not Equal.

$\max(LCS(0,1), LCS(1,0))$

$= \max(0,0) = 0$

The longest subsequence is given by QPQR, Others are PQRR, QPRR.

LCS(1,2) S1[1] = Q, S2[2] = Q equal

= 1 + LCS(0,1)

= 1 + 0 = 1

LCS(1,3) S1[1] = Q
S2[3] = P. Not Equal.

= Max( LCS(0,3), LCS(1,2))

= Max( 0, 1)

= 1

LCS(1,4) = S1[1] = Q
S2[4] = R Not Equal.

= Max (LCS(0,4), LCS(1,3))

= Max (0, 1)

= 1

LCS(1,5) = S1[1] = Q
S2[5] = Q Equal.

= 1 + LCS(0,4)

= 1 + 0

= 1

LCS(1,6) = S1[1] = Q.
S2[6] = R. Not Equal.

= ~~1 + LCS~~ Max (LCS(0,6), LCS(1,5))

= Max(0, 1)

= 1

LCS(1,7) = S1[1] = Q
S2[7] = P Not Equal

Max (LCS(0,7), LCS(1,6)) = Max (0,1) = 1

LCS(2,1) = S1[2] = P.
S2[1] = P.
equal.
1 + LCS(1,0)
= 1+0 = 1.

LCS (2,2)   S1[2] = P.
S2[2] = Q.   Not Equal.
Max ( LCS(1,2) , LCS(2,1))
=> Max(1,1) = 1.

LCS (2,3)   S1[2] = P
S2[3] = P   Equal.
~~Max (LCS~~
1 + LCS (1,2)
= 1+1
= 2.

LCS (2,4)   S1[2] = P
S2[4] = R.   Not Equal.
Max ( LCS(1,4), LCS(2,3))
Max ( 2, 2)
= 2

LCS (2,5)   S1[2] = P
S2 [5] Q.   Not Equal.
Max [LCS(1,5), LCS(2,4)]
Max (1, 2)
= 2.

LCS (2,6)

S1 [2] = P
S2 [6] = R.

Not Equal.

Max (LCS (1,6), LCS(2,5))

Max ( 1, 2)

= 2.

LCS (2,7)

S1[2] = P
S2 [7] = P.

Equal

= 1 + LCS (1, 6)

= 1+ 1

= 2.

Similarly we can also evaluate.

LCS (3,1) = 1
LCS (3,2) = 2
LCS (3,3) = 2
LCS (3,4) = 2
LCS (3,5) = 3
LCS (3,6) = 3
LCS (3,7) = 3

LCS (4,1) = 1
LCS (4,2) = 2
LCS (4,3) = 2
LCS (4,4) = 3
LCS (4,5) = 3
LCS (4,6) = 4
LCS (4,7) = 4

LCS (5,1) = 1
LCS (5,2) = 2
LCS (5,3) = 2
LCS (5,4) = 3
LCS (5,5) = 3
LCS (5,6) = 4
LCS (5,7) = 4

The longest common subsequence can be determined by dynamic programming array by traversing backwards.

The possible LCS's are.

QPQR
PQRR.
QPRR.

# 6.4.4 0/1 KNAPSACK

– Given a set of Items having fixed profits & weights and a knapsack with a fixed capacity the objective here is to select a set of items which can be put/added to the knapsack and the profit is maximized.

→ In other words, we have to select a set of items such that their sum of weights is less than or equal to the capacity of the knapsack and the profit is maximised.

| | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|
| Wt | 10 | 20 | 30 Kgs |
| V | \$60 | \$100 | \$120 |

V ↑ Value

W = 50 Kgs (Capacity of the knapsack)

| | Weight | Value |
|---|---|---|
| $I_1$ | 10 | 60 |
| $I_2$ | 20 | 100 |
| $I_3$ | 30 | 120 |
| $\phi$ | | |
| $I_1, I_2$ | 30 | 160 |
| $\boxed{I_2, I_3}$ | $\boxed{50}$ | $\boxed{220}$ — Best/Optimal solution. |
| $I_1, I_3$ | 40 | 180 |
| $I_1, I_2, I_3$ | 60 > 50 ✗ | 280 |

## ① Optimal Substructure

**Recurrence Relation** :- If $K(n, W)$ represents the maximum profit can be obtained by using a knapsack of capacity W and n items. It can be written using the following recurrence relation.

$$K(n, W) = \begin{cases} K(n-1, W), & \text{if } Wt[n] > W \\ \cancel{\text{Max}(Val[n] + K(n-1, W-Wt[n]), K(n-1, W))}, \cancel{\text{Otherwise.}} \\ \text{Max}\left(Val[n] + K((n-1), (W - Wt[n])), K(n, W-Wt)\right) \\ \cancel{\text{where } Wt[i]}. \end{cases}$$

$Wt[i]$ is the weight of the ith item

$Val[i]$ is the profit of the ith item.

- In the first case the item n is skipped because it cannot be accomodated in the knapsack.
- In the second case ~~the item~~ there are 2 cases.

  a. The Item n is included in the knapsack, that is why its profit is added and its weight is reduced.

  b. The Item n is not included or skipped.

  Maximum of both a, b is considered which ever produces the maximum profit is included.

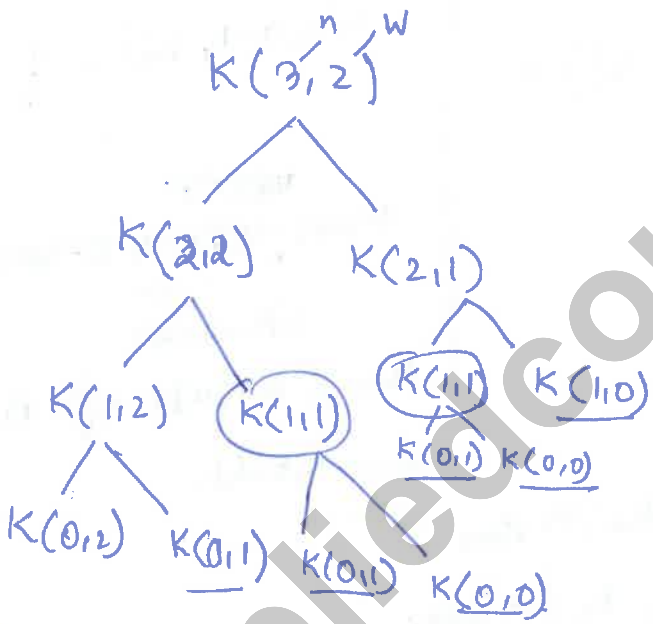
- As the problem can be expressed in forms of smaller problems of the same problem it satisfies the optimal substructure property.

## ② Overlapping Sub problems

$K(n, w)$.

Lets assume $wt[] = \{1, 1, 1\}$ $W = 2$.

$Val[] = \{10, 20, 30\}$




We have repeating sub problems, which is why it satisfies overlapping subproblems property.



Pseudo Code :-

```
KnapSack (W, wt[], val[], n)          // we are having n+1 items
K [n+1][W+1]                          Capacity of Knapsack = W
for(i=0 to n)
{
   ~~for (j=0 to W) # for j=0 to~~
      for(w=0 to W)
      {
         if( i == 0 OR w == 0)
         {
            K[i, w] = 0
         }
```

else if (wt[i-1] <= w)

{

K[i,w] = max (Val[i-1] + K[i-1 + w - wt[i-1]], K[i-1, w])

}

else

{

K[i, w] = K[i-1, w]

}

}

}

return K[n, W]

Time Complexity :-

The outer loop executes (n+1) times

The inner loop executes W+1 times

The statements in the inner for loop can be executed in constant time

∴ The time complexity $O(n*W)$

Space Complexity :-

It takes up an extra space of an array of size $n+1 \times W+1$

∴ Space Complexity = $O(n*W)$

# 64.5 0/1 Knapsack Problem Example.



We know

$$K(n,W) = \begin{cases} K(n-1, W) & \text{if } wt[n] > W \\ \max\{ K(n-1, W), \\ \quad K(n-1, W - wt[n]) + Val[n] \} \end{cases}$$

Let us consider the following example Wt = 7

W = 7 ( capacity of the knapsack).

| item | Wt | Value | W=0 | W=1 | W=2 | W=3 | W=4 | W=5 | W=6 | W=7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 4 | 0 | 1 | 1 | 4 | 5 | 5 | 5 | 5 |
| 3 | 4 | 5 | 0 | 1 | 1 | 4 | 5 | 6 | 6 | 9 |
| 4 | 5 | 7 | 0 | 1 | 1 | 4 | 5 | 7 | 8 | 9 |

$K(1,1) = \max(1 + K(0,0), K(0,1))$

$\max(1+0, 0)$

$= 1$

$K(1,2) = \max(1 + K(0,1), K(0,2))$

$= \max(1+0, 0)$

$= \max(1,0) = 1$

Similarly We Can Calculate All the Other values by applying the recurrence relation.

$$K(1,3) = \text{Max}\{K(n-1,w),\ K(n-1,w-wt[n]) + Val[n]\}$$
$$= \text{Max}\{K(0,3),\ K(0,2)+1\}$$
$$= \text{Max}\{0,1\} = 1$$

$$K(1,4) = \text{Max}\{K(0,4),\ K(0,3)+1\}$$
$$= \text{Max}\{0,1\}$$
$$= 1$$

$K(1,6) = 1$
$K(1,7) = 1$ } – similarly.

Now $n=2$.

$K(2,0) = 0$

$K(2,1) =$ As $wt[2] > w$, $3 > 1$

$K(2,1) = K(1,1)$
$= 1$

$K(2,2) =$ As $wt[2] > 2$, $3 > 2$

$K(2,2) = K(1,2)$
$= 1$

$$K(2,3) = \text{Max}(K(2,2),\ K(1,0) + Val[2])$$
$$= \text{Max}(1,\ 0+4)$$
$$= \underline{4}$$

Similarly we can calculate using the recurrence relation.

$K(2,4) = 5$

$K(2,5) = 5$

$K(2,6) = 5$

$K(2,7) = 5$

$K(3,0) = 0$

$K(3,1) = 1$

$K(3,2) = 1$

$K(3,3) = 4$

$K(3,4) = 5$

$K(3,5) = 6$

$K(3,6) = 6$

$K(3,7) = 9$

$K(4,0) = 0$

$K(4,1) = 1$

$K(4,2) = 1$

$K(4,3) = 4$

$K(4,4) = 5$

$K(4,5) = 7$

$K(4,6) = 8$

$K(4,7) = 9.$

Here the Maximum profit which can be obtained is given by $K(4,7) = 9$.

To know which particular items have been added we need to traverse back the dynamic programming array from $K(4,7)$

$K(4,7)$ here $Wt[4] = 5 < 7$.

$$\text{Max}\{\underbrace{K(3,7)}_{9\ \checkmark},\ \underbrace{K(3,2)+7}_{1+7}\}$$

Which Means Item 4 is not included

Now lets go back from $K(3,7)$

$K(3,7)$ $\quad wt[3] = 4 < 7$

$$= \text{Max}\{\underbrace{K(2,7)}_{5},\ \underbrace{K(2,3)+5}_{4+5=9\ \checkmark}\}$$

Which means Item 3 was added to the Knapsack

Now going back from $K(2,3)$.

$$K(2,3) = \text{Max}\{\cancel{K(1,0)}\ \underbrace{K(1,3)}_{1},\ \underbrace{K(1,0)+4}_{0+4\ \checkmark}\}$$

→ Which Means Item 2 was added to the Knapsack.

→ Now continuing from $K(1,0)$ As Wt is 0 No further Items can be added. $\therefore$ optimal soln $= \{2,3\}$

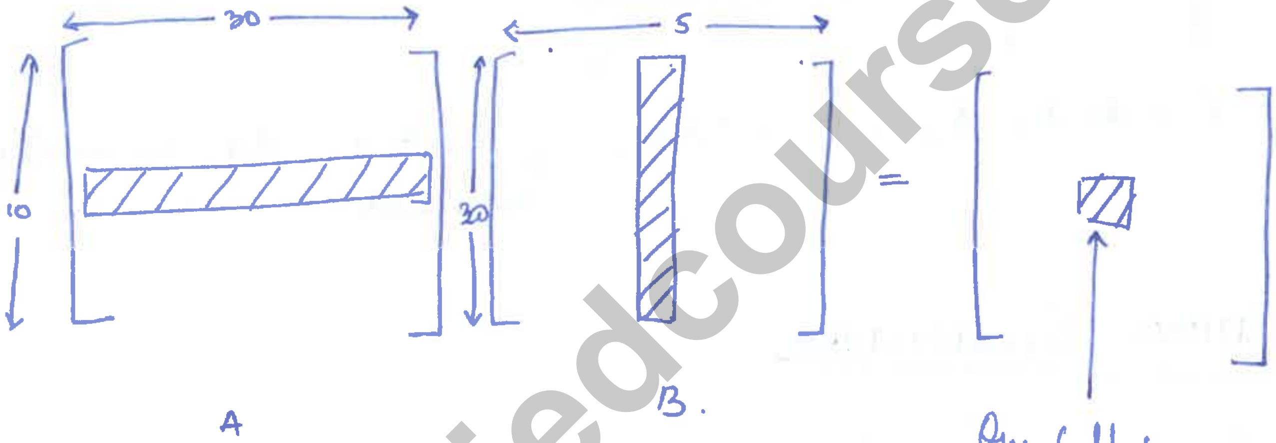
# 64.6. Matrix Chain Multiplication



(1) – Matrix Multiplication is associative in nature i.e. $A.(B.C) = (AB)C$.

If $A_{10\times 3}$ and $B_{30\times 5}$ then $AB = T_{10\times 5}$

No of operations required for the above matrix multiplication is $= 10 \times 30 \times$




One cell corresponds to dot product of 1 row and 1 col of first and second Matrices respectively.



If 3 Matrices are to be multiplied

$ABC = (AB)C = A(BC)$.

$A_{10\times 30}$ $B_{30\times 5}$ $C_{5\times 60}$.

If we Multiply i.e. as $(AB)_{10\times 5}\, C_{5\times 60}$ $= \underline{(10\times 30\times 5)}_{AB} + \underset{AB.C.}{(10\times 5\times 60)}$

$= \underline{4{,}500}$.

If we Multiply using $A_{10\times 30}(BC)_{30\times 60}$ $\Rightarrow (30\times 5\times 60) + (10\times 30\times 60)$

$= 27{,}000$.

No of ways $(N+1)$ matrices can be parenthesized is given by $\frac{(2N)!}{N!\,(N+1)!}$
which is a very large number, if we follow brute force approach then exploring the complete solution space to get the optimal? would be a very huge amount of time which is almost of factorial size.

Important Q 1- What is the optimal parenthesization? the way in which we can get minimum no of operations.

Lets tryout using Dynamic Programming.

$R = A_1 A_2 A_3 \ldots\ldots A_n$. If $A_1, A_2 \ldots A_n$ are multiplication compatable.

## (1) Optimal Substructure

$\{A_i, A_{i+1} \ldots A_j\}$ lets say $M[i,j]$ cost of multiplying $A_i A_{i+1} \cdots A_j$

$M[i,j]$ can be given by the following recurrence relation

$A_i$ has dimensions $P_i \times P_{i+1}$ .... $A_j$ has dimension $P_{j-1} \times P_j$
$A_{i+1}$ $P_{i+1} \times P_{i+2}$
⋮

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i <= k < j} M[i,k] + M[k+1,j] + P_{i-1} P_k P_j \end{cases}$$

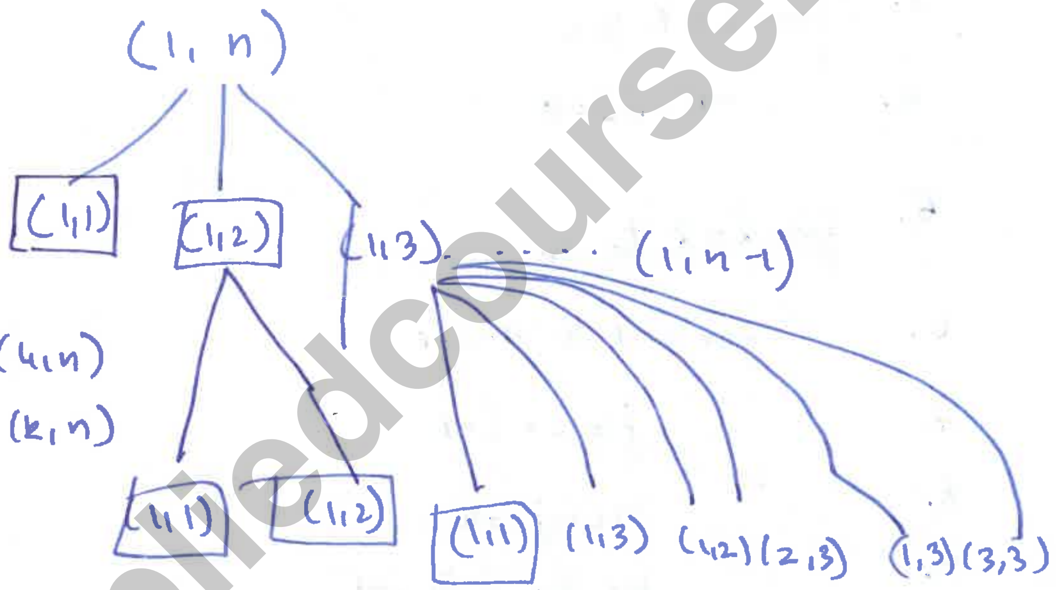
$$\underbrace{(A_i A_{i+1} \ldots A_k)}_{P_i \times P_k} \underbrace{(A_{k+1} \ldots\ldots A_j)}_{P_k \times P_j} = \#\text{ of operations} = P_i \times P_k \times P_j$$

A problem $M[i,j]$ is broken into smaller subproblems

$M[i,k]$ and $M[k,j]$ are used. $\forall\, i < k < j$

Optimal substructure property is satisfied this way.

② Overlapping Subproblems




$(1,n)$ — $(1,3)(4,n)$
$(1,k)(k,n)$

As we can see we have sub-problems which are overlapping in nature

→ By simple recursion if we do not make use of dynamic programming then the time complexity = ?

The no of ways of parenthesizing $(n+1)$ matrices is given by

Catalan Number $C_n = \left({}^{2n}C_n\right) / (n+1)$ or $\dfrac{2n!}{n! \times (n+1)!}$

$C_n \sim \dfrac{4^n}{n^{3/2} \ast \sqrt{\pi}} = \Omega(2^n)$ ← exponential time algorithm

# Bottom Up Dynamic Programming Approach



From CLRS.

## MATRIX-CHAIN-ORDER(p)

1. $n = p.length - 1$
2. let $m[1 \dots n, 1 \dots n]$ and $s[1 \dots (n-1), 1 \dots (n-1)]$ be new tables
3. for $i = 1$ to $n$
4. $\quad m[i,i] = 0$
6. for $l = 2$ to $n$
6. $\quad$ for $i = 1$ to $n - l + 1$
7. $\quad\quad j = i + l - 1$
8. $\quad\quad m[i,j] = \infty$
9. $\quad\quad$ for $k = i$ to $j-1$
10. $\quad\quad\quad q = m[i,k] + m[k+1,j] + (p_{i-1} * p_k * p_j)$
11. $\quad\quad\quad$ if $q < m[i,j]$
12. $\quad\quad\quad\quad m[i,j] = q$.
13. $\quad\quad\quad\quad s[i,j] = k$.
14. return $m$ and $s$.

Note :- $p$ is input array of size $n+1$ which consists of all the dimensions of the $n$ arrays.

$m$ is the array which ~~stat~~ stores the minimum no of operations required to multiply the matrices i.e. $m[i,j]$ stores the minimum no of operations required to multiply the matrices $A_i A_{i+1} \dots A_{j-1} A_j$

→ Array s stores the optimal split point. If optimal split pt for ~~$A_1 A_2 A_3$~~ $A_i A_{i+1} \dots A_{j-1} A_j$ is at k, then k is stored at s[i, j].

## Time Complexity

- The loop corresponding to l will run almost n times (line 5)
- The loop corresponding to i will run atmost n times (line 6)
- The loop corresponding to k which is the break point will also run n times (line 9).
- Therefore the time complexity is $O(n^3)$ polynomial time complexity
- By using Dynamic Programming we are able to reduce a problem from exponential to polynomial time.

→ This problem has a lot of applications and significance in scientific computing.

# 64.7 Subset Sum Problem



Problem :- Given a set of non negative numbers, and a given sum, are there items in the set such that their sum is equal to SUM.

I/p Set [] = {3, 34, 4, 12, 5, 2}
SUM = 9.

(Q) are there items in the set such that sum of the items is equal to SUM.? True / False Result

{4, 5} ⇒ 4+5 = SUM = 9 True.

## Brute Force Approach

Set of n numbers is given.

The number of subsets possible are $2^n$

Let's generate all the possible subsets of n and find out if ∃ a subset whose sum of elements = SUM.

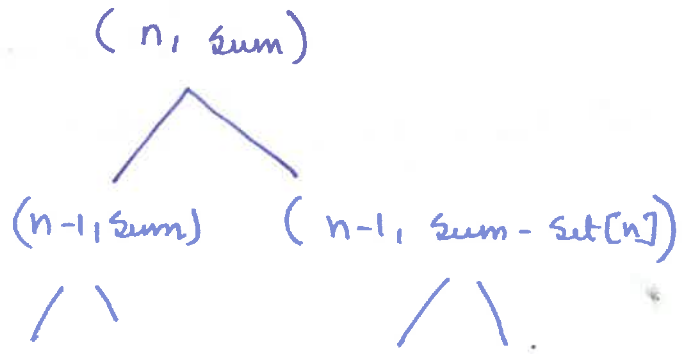
- This approach has an exponential time complexity $O(2^n)$

## Recursion (Optimal substructure)

$$S(n, SUM) = \begin{cases} \text{FALSE}, & \text{if } n=0 \text{ \& } SUM>0 \\ \text{TRUE}, & \text{if } SUM=0 \\ \underbrace{S(n-1, SUM)}_{\text{not using the nth item}} \text{ OR } \underbrace{S(n-1, SUM-set[n])}_{\text{using the nth item}} \end{cases}$$

(first two cases) - Base Case.

## Overlapping Subproblems




- There will be cases when we have 2 identical subproblems occuring in the recursion tree, in such cases the overlapping subproblems property will be satisfied.

- There may also be cases in this problem where there are no overlapping subproblems. In such cases DP reduces to simple recursion.

### DP Algorithm

```
isSubsetSum( set[], int n , int sum)
{
    subsetSum [n+1][sum+1] // boolean array value of subset of
                           // subset [i][j] will be true if
                           // there exist a subset of set [0...j-1]
                           // with sum = i

    for i=0 upto n   // if sum=0 then it is always true.
    {
        subset [i][0] = true;
    }
```

```
for (i=0; i <= sum; i++)      // if sum is non zero and the
{                              // set is ~~non~~ empty answer is
    subset[0][i] = false;      // false.
}

for (i = 1 upto n)
{
    for (j = 1; upto sum)
    {
        if (j < set[i-1])
        {
            subset[i][j] = subset[i-1][j];
        }
        if (j >= set[i-1])
        {
            subset[i][j] = (subset[i-1][j] || subset[i-1][j-set[i-1]])
        }
    }
}
```

## Time Complexity

- First two for loops iterate $n$ times their time complexity $O(n)$
- Nested for loop :-
  - Outer for loop executes $n$ times
  - Inner for loop executes sum times

    ~~The time complexity $O(n \times sum)$~~

  - Within the inner for loop the if-else statements can be executed in constant time.
  - The time complexity $= O(n \times sum)$.

- Total time complexity of DP Algo $O(n \times sum + n)$

$= O(n * sum)$.

Brute force time complexity $O(2^n)$

if $sum = 2^n$ or $sum > 2^n$ then in such a case DP performs worse than DP.

- The DP is polynomial in $n$ and sum.
- The brute force is exponential in $n$ (independent of sum).

* So if $sum < 2^n$ its better to go with the DP. Otherwise if ~~$sum > 2^n$~~ $sum > 2^n$ it is better to go with brute force approach

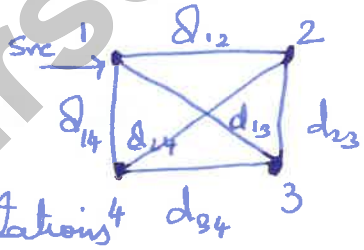
(Very important case in DP algorithms).

# 64.8 TRAVELLING SALESMAN PROBLEM

- Problem in Graph Theory.
- Given n cities which are represented by n vertices we need to find the minimal path such that all the cities/vertices are visited once starting from the source vertex and returning back to the source vertex.

Main Idea

- Overall path length/distance should be minimal.




BRUTE FORCE All permutation of all the cities is tried

now for n cities we have n! permutations

n! permutations are possible

$O(n!)$ — generate the permutations and compute the path cost in case of each permutation and pick the minimal among all.

Recursive Solution

Let the vertex set be $V = \{1, 2, 3, \ldots, n\}$.

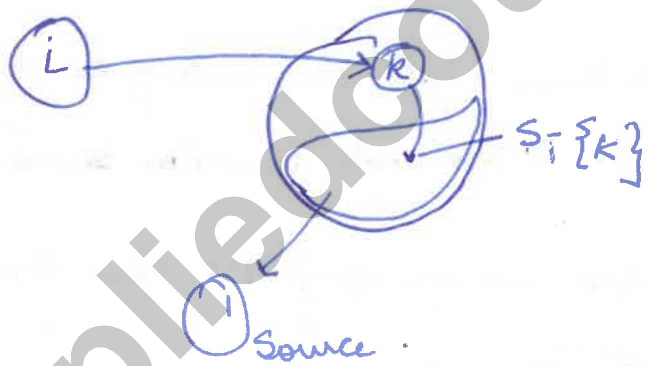
If Source is vertex 1

$d_{ij}$ = represents distance b/w vertex i and j

$S \subseteq V \quad S = \{2, 3, 4 \ldots n\} = V - \{1\}$ (excluding the source vertex)

$$TSP(i,S) = \begin{cases} d_{i1} & \text{if } S \neq \phi \quad \text{Base Case.} \\ \min\limits_{k \in S} \left[ d_{ik} + TSP(k, S-\{k\}) \right] & \text{Otherwise.} \end{cases}$$

Minimum cost ~~vertex~~ path from $i^{th}$ vertex to every vertex in S and back to the source (vertex 1 in our case).




Min Cost from i to k + Min Path of vertices in S - {k} back to the source 1.

Recursion Tree

n elements ← TSP(1, {2,3,4}) → n

(n-1) subproblems. TSP(2, {3,4}) TSP(3, {2,4}) TSP(4, {2,3}) →

(n-1)(n-2) s.p's TSP(3, {4}) TSP(4, {3}) TSP(2, {4}) TSP(4, {2}) TSP(2, {3}) ~~TSP(4, {2})~~

(n-1)(n-2)(n-3)...1 TSP(4, { }) TSP(3, { }) TSP(4, { }) TSP(2, { })

TSP(3, { }) TSP(4, {2})

TSP(2, { })

Total no of sub problems = $(n-1) + (n-1)(n-2) + (n-1)(n-2)(n-3)$

$+ \dots \dots (n-1)(n-2) \dots 1$

$= O(n) + O(n^2) + O(n^3) + \dots O(n^{n-1})$

$= O(n^n)$

- The Brute force algorithm $O(n!)$

By using stirlings approximation we know $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

so we now have $O(n!) \simeq O(n^n)$

So now we have that the brute force approach is almost equvalent to ~~DP~~ Approach Recursion based approach.

→ However when we are using DP then there will be some overlapping subproblems (As we can observe in the recursion tree on the previous page).

→ If we write the algorithm of TSP using Bottom up approach (i.e by using tabulation) & by making use of the recursive formula this is known as the Held Karp Algo which has time complexity $O(2^n n^2)$ and the space required $= O(2^n \cdot n)$.

Even though there is an improvment $n^n > 2^n$

→ TSP cannot be reduced more to faster than exponential time complexity

→ It belongs to complexity ~~class~~ class of problems which are NP-HARD



## Bottom-Up-DP Algorithm (tabulation) for TSP

function TSP(G, n)
{
for (k = 2 upto n!-)
{
$C(\{k\}, k) = d_{1,k}$ // cost from source to vertex k is updated
}
for (s = 2 upto n-1)
{
for (all $S \subseteq \{2, 3, \dots n\}$, $|S| = s$) do
{
for (all k ∈ S) do.
{
$C(S, k) = \min_{\substack{m \neq k \\ m \in S}} [C(S - \{k\}, m) + d_{m,k}]$
}
}
}
optimal :- $\min_{k \neq 1} [C(\{2, 3, \dots n\}, k) + d_{k,1}]$
} return optimal path

→ Example





Let us take the following example.

$$D = \begin{bmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{bmatrix}$$

Let have the following function description

→ $TSP(x, S)$ :- Starting from vertex 1 (source), path of min cost that ends at vertex $x$, passing through all the vertices in set $S$ exactly once.

→ $d_{xy}$ - edge cost from $x$ to $y$.

→ $p(x, S)$ :- the second to last vertex or the source vertex from set $S$. This is used for constructing the TSP path at the ~~End~~ end.



Initially $k=0$ we have $S$ as Null Set.

$TSP(2, \phi) = d_{21} = 1$

$TSP(3, \phi) = d_{31} = 15$

$TSP(4, \phi) = d_{41} = 6.$

considering sets of size 1 element.

Set $S = \{2\}$

$$TSP(3, \{2\}) = d_{32} + TSP(2, \phi) = d_{32} + d_{21}$$
$$= 7 + 1$$
$$= 8 \qquad p(3, \{2\}) = 2.$$

$$TSP(4, \{2\}) = d_{42} + TSP(2, \phi) = d_{42} + d_{21}$$
$$= 3 + 1$$
$$= 4$$
$$p(4, \{2\}) = 2$$

Set $S = \{3\}$

$$TSP(2, \{3\}) = d_{23} + TSP(3, \phi) = d_{23} + d_{31} = 6 + 15 = 21$$
$$p(2, \{3\}) = 3$$

$$TSP(4, \{3\}) = d_{43} + TSP(3, \phi) = d_{43} + d_{31} = 12 + 15 = 27$$
$$p(4, \{3\}) = 3.$$

Set $S = \{4\}$

$$g(2, \{4\}) = C_{24} + g(4, \phi) = C_{24} + C_{41} = 4 + 6 = 10$$

$$* g(3, \{4\}) = C_{34} + g(4, \phi) = C_{34} + C_{41} = 8 + 6 = 14$$

Now considering all possible subsets of 2 elements

$S = \{2,3\}$.

$TSP(4,\{2,3\}) = \min\{d_{42} + TSP(2,\{3\}), c_{43} + TSP(3,\{2\})\}$

$= \min\{3+21, 12+8\}$

$= \min\{24, 20\} = 20$. $\quad P(4,\{2,3\}) = 3$

Set $S = \{2,4\}$

$TSP(3,\{2,4\}) = \min\{d_{32} + TSP(2,\{4\}), d_{34} + TSP(4,\{2\})\}$

$= \min\{7+10, 8+4\} = \min(17,12) = 12$

$P(3,\{2,4\}) = 4$

Set $S = \{3,4\}$

$TSP(2,\{3,4\})$

$= \min\{d_{23} + TSP(3,\{4\}), d_{24} + TSP(4,\{3\})\}$

$= \min\{6+14, 4+27\} = \min\{20,30\} = 20$

$P(2,\{3,4\}) = 3$.

Now considering subset of size $= 3$

$TSP(1,\{2,3,4\}) = \min\{d_{12} + TSP(2,\{3,4\}), d_{13} + TSP(3,\{2,4\}), d_{14} + TSP(4,\{2,3\})\}$

$= \min\{2+20, 9+12, 10+20\} = \min\{22, 21, 30\}$

$= 21$

$P(1,\{2,3,4\}) = 3$

Now to get the optimal path we need

Successor of node 1 $p(1,\{2,3,4\}) = 3$.

Successor of node 3 $p(3,\{2,4\}) = 4$.

Succ. of node 4 $p(4,\{2\}) = 2$.

∴ Now the optimal path is given by $1 \to 3 \to 4 \to 2 \to 1$

---

# 64.9 Bellman Ford Algorithm As DP

PROBLEM :- Single source shortest path, given a single source in in a Graph G we need to calculate the cost of the shortest path to every other vertex in the graph G.

→ Bellman ford Algorithm returns the single source shortest paths even for graphs which have −ve edges, however it returns FALSE in case the graph ~~does not~~ contains a −ve weight cycle and it does not return the shortest paths in such a case.

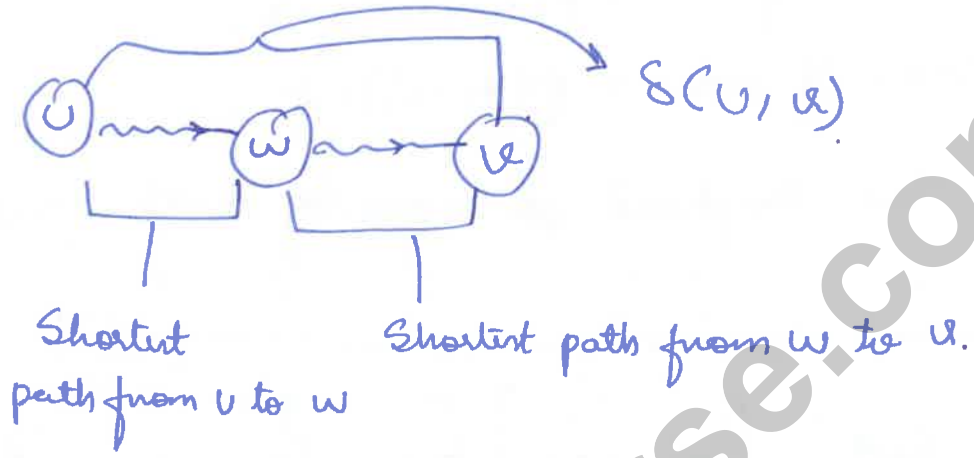
→ Named after Richard Bellman who is the creator of Dynamic programming and Lester Ford, Jr. both of them published it in 1958 and 1956 respectively.

How is the Bellman Ford Algorithm posed as a DP problem?

let $\delta(u,v) = \begin{cases} \text{distance of shortest path from } u \to v \\ \infty \text{ if no path exists from } u \to v. \end{cases}$

## OPTIMAL SUBSTRUCTURE :-

— Subparts of shortest paths are the shortest paths.




**Proof by contradiction**

If we have another path from u to w which is shorter than the existing path from u to w. ~~If~~ Let us assume.

- If the above assumption is true then, it would have been ~~to~~ a part of the u-v shortest path which is not true, hence the original u - w path is only the shortest path from u-w, this is proof by contradiction.

→ Bellman-Ford ( G, w, s).

```
INITIALIZE-SINGLE-SOURCE (G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX (u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        Return FALSE.
return TRUE.
```

## → Central Idea of Bellman Ford Algorithm :-

→ Shortest path between ~~and~~ any two vertices (u, v) has atmost |V-1| edges.

→ The Bellman ford algorithm first ~~calucula~~ calculates shortest paths of length 1, 2, 3, . . . . (|V|-1) in a bottom up manner. i.e. while calculating shortest path of length $i$ the results of shortest paths of length $(i-1)$ are reused.

– The Algorithm from lines 2-4 the ~~ed~~ loop for value of $i = 1$ to $(|V|-1)$, in the $i$ th iteration ~~the edge edges of length~~ the paths of length atmost $i$ edges is determined which are the shortest in terms of cost or in other words shortest paths of at most $i$ edges is determined in the $i$ th iteration.

– Lines 5 to 7 are for determining -ve weight cycles.

## Overlapping Sub problems in Bellman Ford

Consider the following example.


In order to calculate the shortest path from $V_1$ to $V_4$ we would reuse the shortest path from $V_1$ to $V_2$ and $V_2$ to $V_4$ and also in order to determine the length of the shortest path from $V_2$ to $V_4$ we need to reuse the shortest paths from $V_2$ to $V_3$ and $V_3$ to $V_4$. We have overlapping subproblems here.

---

## 64.10 Floyd Warshall Algorithm As Dynamic Programming

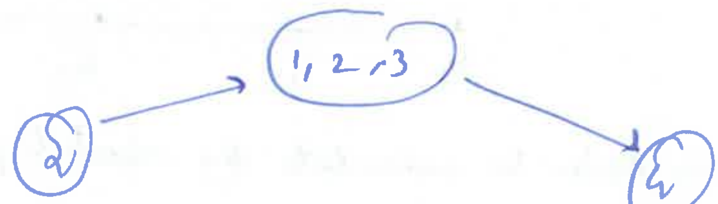
→ All pairs shortest path problem.

**Recursive Formulation :-**

V is the set of vertices $\{1, 2, 3, \ldots n\}$

$d_{ij}^{(k)}$ = distance of the shortest path from vertex $i$ to vertex $j$ with vertices $\{1, 2, \ldots k\}$ as possible intermediate vertices.

For eg $d_{24}^{(3)}$ = represents shortest path b/w 2 and 4 by using vertices $\{1, 2, 3\}$ as possible intermediate vertices

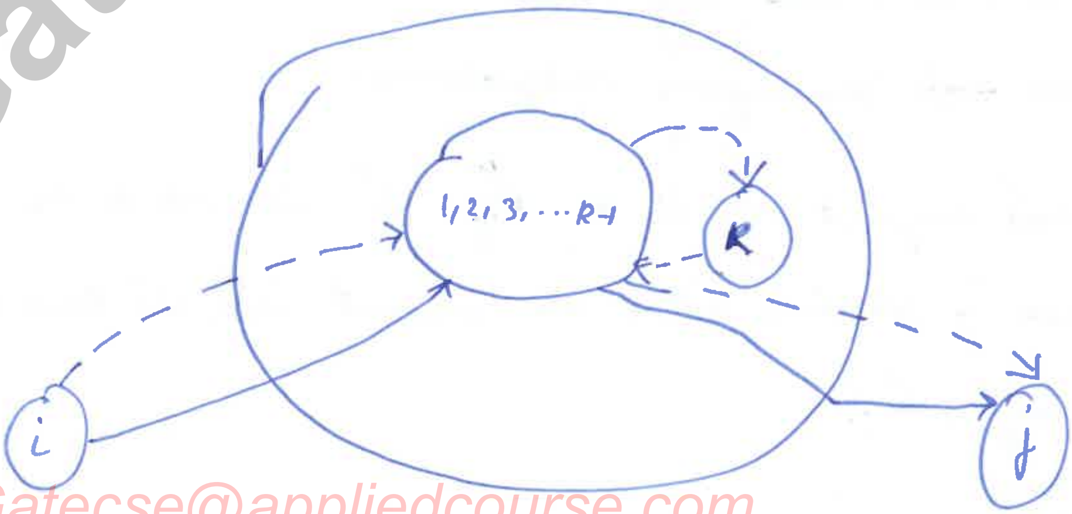


Ph: 844-844-0102

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & , \text{ if } k=0. \\ \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

- If $k=0$ it means that no intermediate vertices are allowed in such a case $d_{ij} = w_{ij}$ ( ∵ weight of the edge between $i$ and $j$).

- If $k \geq 1$ here we have two cases.
  1. $d_{ij}^{(k-1)}$ is when we do not make use of the $k$th vertex
  2. $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ is when we make use of the $k$th vertex the shortest path from $i$ to $k$ and $k$ to $j$ by using the $(k-1)$ vertices are used.





Mail: Gatecse@appliedcourse.com

FLOYD-WARSHALL(W)

1. $n = W.rows$
2. $D^{(0)} = W$
3. for $k = 1$ to $n$
4. let $D^{k} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5. for $i = 1$ to $n$
6. for $j = 1$ to $n$.
7. $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8. return $D^{(n)}$.

→ Initially $D^0$ is intialised by W

→ $D^1$ is calculated using $D^0$

$D^2$ is calculated using $D^1$

$D^3$ is calculated using $D^2$

$\vdots$

$D^i$ is calculated ~~used~~ using $D^{i-1}$

we have overlapping subproblems.

Our final required result is $D^n$ in which the all pairs of shortest paths are present and all the n vertices are being used.

## 64.11 SOLVED GATE 2016

1. The Floyd Warshall algorithm for all pairs shortest paths is computation is based on.

   A. Greedy paradigm.

   B. Divide and Conquer paradigm.

   C. Dynamic Programming paradigm.

   D. ~~neig~~ neither greedy, nor divide and conquer nor dynamic programming paradigm.

Ans :- C. Floyd Warshall is an example of Bottom up Dynamic Programming.

## 64.12. SOLVED PROBLEM GATE 2015

LIST - I

A. Prim's algorithm for minimum spanning tree

B. Floyd-Warshall algorithms for all pairs shortest path

C. Merge Sort

D. Hamiltonian Circuit.

LIST - II

1. Backtracking
2. Greedy Method.
3. Dynamic Programming
4. Divide & Conquer.

Options

| | A | B | C | D. |
|---|---|---|---|---|
| (a) | 3 | 2 | 4 | 1 |
| (b) | 1 | 2 | 4 | 3 |
| (c) | 2 | 3 | 4 | 1 |
| (d) | 2 | 1 | 3 | 4 |

Soln :-

B. We know that Floyd Warshall is a DP Algorithm.

B – 3.

C. Merge Sort is a divide and conquer algorithm.

C – 4

From this we can choose option c.

A: Prims is a greedy algorithm for constructing MST.

A – 2.

D. Hamiltonian cycle is constructed by using backtracking.

D – 1

∴ option c is correct.

# 64.13. Solved Problem GATE 2011



3. Four Matrices M1, M2, M3, and M4 of dimensions $p \times q$, $q \times r$, $r \times s$, $s \times t$ respectively can be multiplied in several ways with different number of scalar multiplications. For example

$((M_1 \times M_2) \times (M_3 \times M_4))$ would require $pqr + rst + prt$.

When multiplied as $(((M_1 \times M_2) \times M_3) \times M_4)$ the total no of scalar multiplications required is $pqr + prs + pst$.

If $p = 10$, $q = 100$, $r = 20$, $s = 5$ and $t = 80$, then the total no of scalar multiplications needed is ?

(A) 248000
(B) 44000
(C) 19000
(D) 25000



Solution :-

We need to find the optimal number of scalar multiplications required for this multiplication.

We have the Recurrence Relation

$$M[i,j] = \begin{cases} 0, & \text{if } i = j \\ \min\limits_{i <= k < j} M[i,k] + M[k+1,j] + P_{i-1} \times P_k P_j & \text{if } i < j \end{cases}$$

$P_0 = 10,\ P_1 = 100,\ P_2 = 20,\ P_3 = 5,\ P_4 = 80.$

Size 0

$M[1,1] = M[2,2] = M[3,3] = M[4,4] = 0$ Base Case.

$M[1,2] = P_0 \times P_1 \times P_2 = 10 \times 100 \times 20 = 20{,}000$

$M[2,3] = P_1 \times P_2 \times P_3 = 100 \times 20 \times 5 = 10{,}000$

$M[3,4] = P_2 \times P_3 \times P_4 = 20 \times 5 \times 80 = 8{,}000$

Size 1

$$M[1,3] = \text{Min} \left\{ \begin{array}{l} M[1,1] + M[2,3] + (10 \times 100 \times 5), \\ M[1,2] + M[3,3] + (10 \times 20 \times 5) \end{array} \right\}$$

$= 15{,}000.$

$$M[2,4] = \text{Min} \left\{ \begin{array}{l} M[2,2] + M[3,4] + (100 \times 20 \times 80), \\ M[2,3] + M[4,4] + (100 \times 5 \times 80) \end{array} \right\}$$

Size 2.

$= 50{,}000.$



$$M[1,4] = \text{Min} \left\{ \begin{array}{l} M[1,1] + M[2,4] + (10 \times 100 \times 80), \\ M[1,2] + M[3,4] + (10 \times 20 \times 80), \\ M[1,3] + M[4,4] + (10 \times 5 \times 80) \end{array} \right\}$$



$= \underline{19{,}000}$ option C

# GREEDY ALGORITHMS

## 66.1 Greedy Algorithms :- Fractional Knapsack.

- Greedy Algorithm :- Another popular algorithm design strategy like dynamic programming.

- Example problem Fractional knapsack problem.
  - In 0/1 Knapsack we can either add the item completely or leave the item completely.
  - In the fractional knapsack problem we can use fraction of the items.

Consider the following example instance

| Items | Value | Weight |
|---|---|---|
| 1 | 60 | 10 |
| 2 | 100 | 20 |
| 3 | 120 | 30 |

W = Capacity of the knapsack = 50

If the problem is 0/1 Knapsack the solution would be

$\{2,3\} \rightarrow 50 <= 50$



Value $\{2,3\} = 100 + 120 = 220$

If the above problem is a fractional knapsack problem.

Lets take Item 1 complitly 1 — 10 — 60
then Item 2 complitely. 2 — 20 — 100
We are left with 20 capacity
lets take 2/3rd of Item 3 3 — 20 — $\frac{2}{3} \times 120 = 80$

Profit of above selection $= 1 + 2 + \frac{2}{3}$ of 3
$= 60 + 100 + 80$
$= \underline{240}$

- The above problem is an example of an optimization problem in which we try to maximize the total profit of items in the knapsack. this is known as the objective or objective function, the condition on the above objective function ~~is~~ ~~that~~ are as follows:-

(i) We can pick fraction of any item.
(ii) The total weight of items we pick must be <= W.

The above two mentioned conditions are known as constraints.

→ Any optimization problem has an objective function and constraints which have to be fulfilled or satisfied.

How can we get the optimal solution for the fractional knapsack problem?

| Items | Value | Wt | Val/Wt |
|---|---|---|---|
| 1 | 60 | 10 | 60/10 = 6. |
| 2 | 100 | 20 | 5 |
| 3 | 120 | 30 | 4 |

→ We calculate the value/Wt of each item

→. Then we choose the items greedily based on the value/Wt which means we select the item having maximum value of value/wt first and then in decreasing order of value/wt.

1. First choose item 1
2. Then choose item 2.
3. Then choose item 3 (But as the remaining wt is 20 2/3 rd of item 3 is chosen).

Time complexity analysis for fractional knapsack problem

1. Calculate the value/wt for each item — $O(n)$ time, $O(n)$ space.
2. Sort the items by Val/wt - $O(n \log n)$ time
3. Start picking the items greedily till the knapsack is full - $O(n)$ time

Total time complexity $= O(n + n\log n + n)$

$= O(n\log n)$. & space complexity $O(n)$.

As in dynamic programming, we have the properties optimal substructure and overlapping subproblems which need to be satisfied by a problem for that problem to be solvable by dynamic programming.

For a problem to be solvable by using the ~~Bxx~~ Greedy strategy it should satisfy the following properties

1. Optimal Substructure :- An optimal solution to the problem will contain optimal solution to subproblems. We can define the problem recursively in terms of smaller subproblems.

2. Greedy Choice Property :- At every step if we greedly pick the solution we would get the optimal solution finally.

In case of fractional knapsack problem.

Optimal Substructure :- After adding item 1 to the knapsack, the problem can be expressed as finding the best knapsack from items 2, 3, which is a subproblem of the orignal problem.

Greedy Choice property :- At each step we keep choosing the item which is having maximum value of value/wt or in other words we are choosing the item greedyly.

Note → Every real world problem need not satisfy the greedy choice property. But if it satisfies then that problem can be solved using greedy approach.

- For the 0/1 Knapsack the greedy choice property is not satisfied.
- For the fractional knapsack problem it is satisfied.

---

## 66.2 Solved Problem : Gate 2018

Consider the weights and values of the items listed below. Note that there is only one unit of each.

| Item number | Weight (in Kgs) | Value (In Rupees) |
|---|---|---|
| 1 | 10 | 60 |
| 2 | 7 | 28 |
| 3 | 4 | 20 |
| 4 | 2 | 24 |

The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover no item may be split. The total value of items picked by an optimal algorithm is denoted by $V_{opt}$. A greedy algorithm sorts the items by their value to weight ratios in decreasing order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is $V_{greedy}$. The value of $V_{opt} - V_{greedy}$ is ____.

A. 16.

B. 8

C. 44

D. 60.

W = 11

Optimal solution can be obtained by DP or bruteforce approach.

If we take

Item 1 — 60 — wt - 10

Item {2+3} — 28+20 = 48 — wt = 7+4 = 11

Item {3+4} — 20 + 24 = 44 — wt = 4+2 = 6

Item {2+3+4} — wt = 7+4+2 = 13 > 11 wt constraint not satisfied

Best possible / optimal soln is Item 1 — Profit — {60}

Now coming to Greedy solution, calculate Value/wt.

| Item Number | Weight | Value | Value/wt |
|---|---|---|---|
| 1 | 10 | 60 | 6 |
| 2 | 7 | 28 | 4 |
| 3 | 4 | 20 | 5 |
| 4 | 2 | 24 | 12 |

First add item 4 —— 4 — 2 (wt) —— 24 (profit)

Add item 1 —— 1 — 10 (wt) (wt constraint not satisfied)
× discard item 1

Add item 3 —— 3 — 4 (wt) —— 20 profit.

Add item 2 —— 2 — 7 (wt) - wt constraint not satisfied
× discard item 2.

∴ Items in Knapsack = {4, 3}, Profit = 24 + 20 = 44 $V_{greedy} = 44$
Wt = 2+4 = 6

$$\text{Value} = V_{opt} - V_{Val} = 60 - 44$$
$$= \underline{\underline{16}} \quad \text{A option}$$

---

# 66.3 HUFFMAN CODING FOR DATA COMPRESSION

- Strategy for data compression.
- Lets assume the following sample text

| text | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Frequency | 5 | 25 | 7 | 15 | 4 | 12 |

$= 68$ # characters in the text.

If we encode each of the characters using binary encoding

For n characters we need 3 bits $\Rightarrow 2^3 = 8$ we can represent 8 characters

2 bits $\rightarrow 2^2 = 4$ characters.

| 00 | 01 | 10 | 11 |
|---|---|---|---|
| A | B | C | D. |

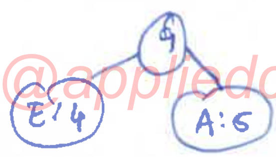
Total space required for 68 characters $= 68 \times 3$ bits/character

$= 204$ bits.

Q) Can we represent using fewer number of bits?

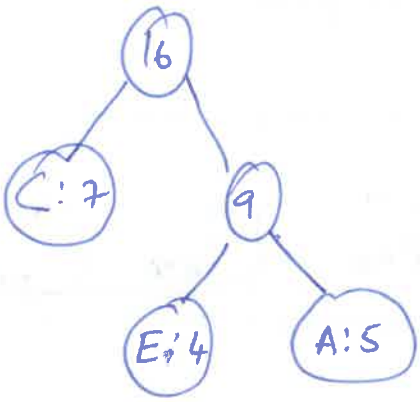
Idea:- Greedily pick the characters that occurs less frequently.

(1) Choose A and E and construct a binary tree



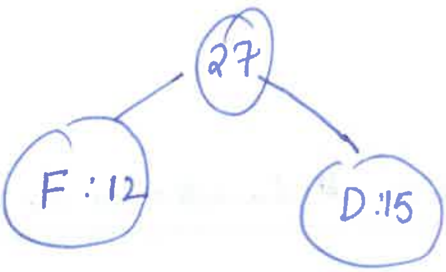
B:25 C:7 D:15 F:12 EA:9

2



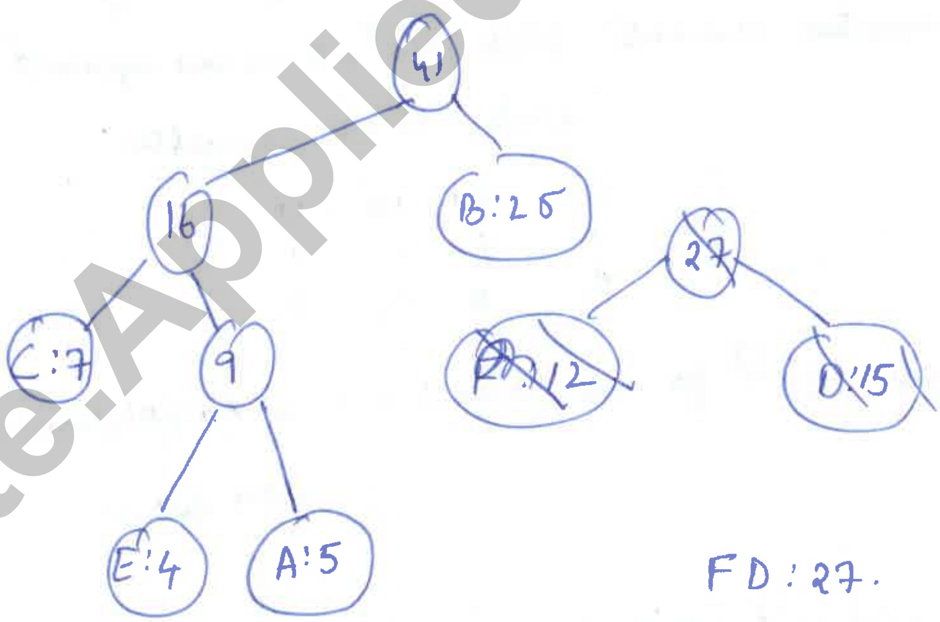

F: 12 ✓
D: 15 ✓
CEA: 16
B: 25

3

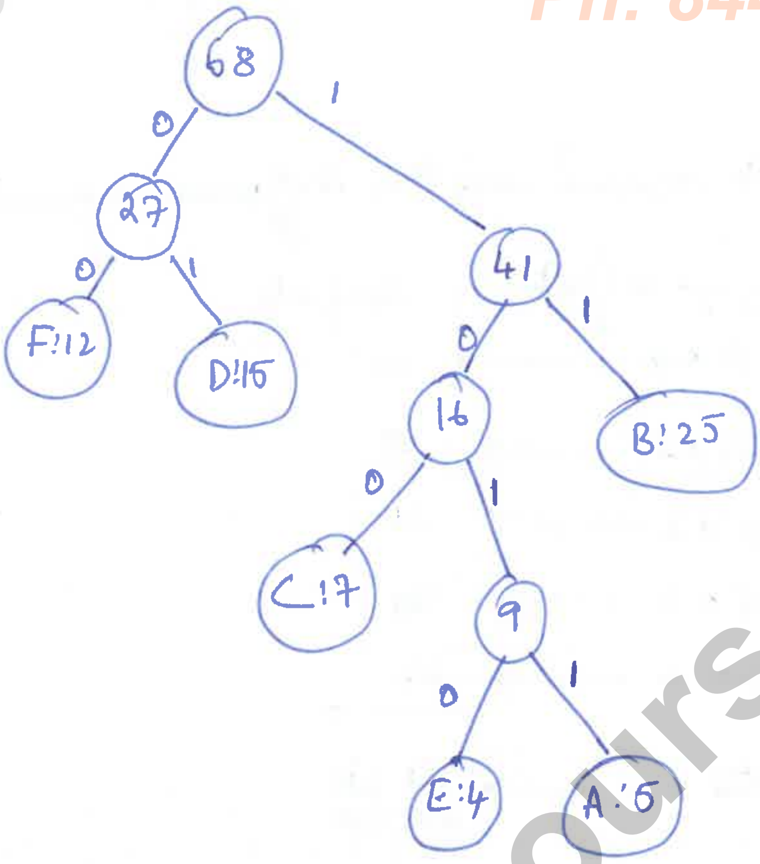



FD: 27
CEA: 16 ✓
B: 25 ✓

4




FD: 27.
CEAB:- 41.




The above is the huffman tree.

- For every node on the left child we label it as 0 and for the right child we label it as 1.

- The Huffman code for any character is obtained by traversing from root to the ~~lef~~ leaf node of that character.

Now Huffman code for

| | Frequency |
|---|---|
| F : 00 | 12 |
| D : 01 | 15 |
| C : 100 | 7 |
| B : 11 | 25 |
| E : 1010 | 4 |
| A : 1111 | 5 |

- If we observe the items which are more frequent are given shorter encoding and the items which are less frequent are assigned longer codes.

This is ~~An~~ also known as Variable length representation.

- As each character is not represented using a fixed/constant/same number of bits.

→ Total no of bits required using this Huffman representation

| | freq × no of bits | = No of bits |
|---|---|---|
| A → | 5 × 4 → | 20 |
| B → | 25 × 2 → | 50 |
| C → | 7 × 3 → | 21 |
| D → | 15 × 2 → | 30 |
| E → | 12 × 2 → | 24 |
| | Total → | 161 bits |

From binary representation (204 bits) we have gained improvement (161 bits).

## Time Complexity

- At each step we are choosing the 2 least frequent characters / set of ~~characters~~ characters and combining it back.

initially we have n items

then (n-1) items ~~Above we can~~

then (n-2) items

⋮

upto we have 1 item.

which means we repeat the process n times n times we pick 2 least freq characters/sets and combine them.

- If an array is used ~~for~~ extracting the 2 least would require $O(n)$ time.

- If we use a min heap extracting the least 2 items would require $O(\log n)$ time.

$\therefore$ The time complexity $= \underline{O(n \times \log n)}$ (using a min heap).

Space Complexity when using a min heap $\underline{O(n)}$.

— Let us consider an example text ABBCD.

the huffman code is

1101 11 11 10001

This can be obtained by simply replacing the huffman ~~bit~~ code for each character.

— In order to decode one must start from the left side of the huffman code and traverse the tree according to the code until a leaf node is reached, once a leaf is encountered the code encountered so far can be replaced by the leaf node character and then we should restart from the root again to decode the next character, this process has to be repeated till the complete 1 string is completed.

On decoding we will get back ABBCD.

Ph: 844-844-0102

— How can we show that it problem that can be solved using the greedy strategy?

ⓐ Greedy Choice Property :- At each step we choose the least frequent character/a set of characters

ⓑ Optimal Substructure :- Every subtree of huffman tree defines an optimal subtree.

Since the above 2 properties are satisfied, the huffman encoding is a greedy design strategy.

— The huffman encoding is an example of ~~the~~ lossless encoding (because there is no information loss).

— Image → jpeg → Compression format (Fourier) → is an example of lossy compression.

↑ Raw Image

---

## 66.4 Solved Problem Gate 2017

A message is made up entirely of characters from the set $X = \{P, Q, R, S, T\}$

| Character | Probability |
|---|---|
| P | 0.22 |
| Q | 0.34 |
| R | 0.17 |
| S | 0.19 |
| T | 0.08 |
| Total | 1.00 |

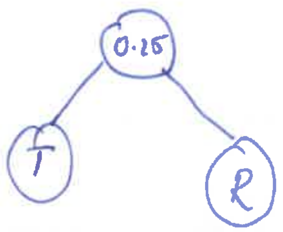

Mail: Gatecse@appliedcourse.com

A message of 100 characters is encoded using Huffman encoding. The expected length of the encoded message is ______.



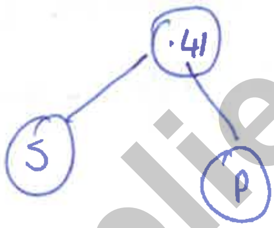
A. 225
B. 226
C. 227
D. 228

1



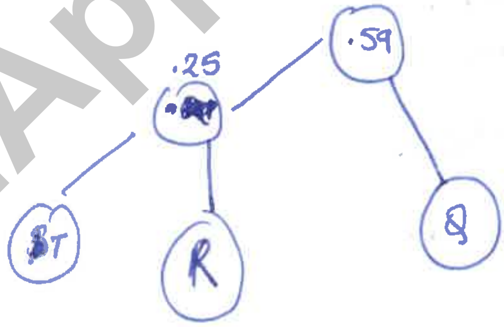
| | |
|---|---|
| P | 0.22 |
| TR | 0.25 |
| Q | 0.34 |
| S | 0.19 |

2



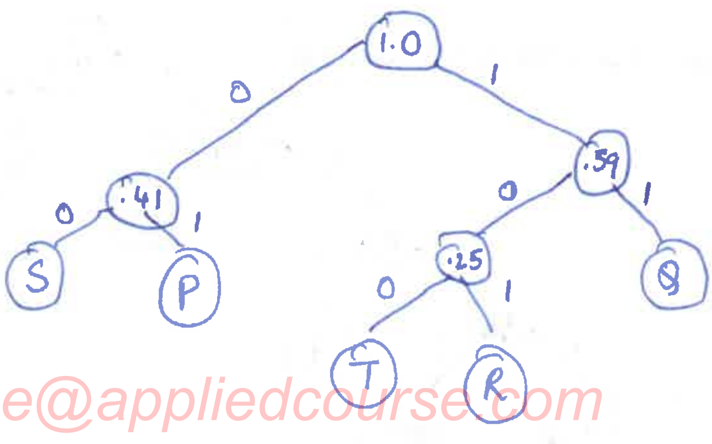
| | |
|---|---|
| PS | 0.41 |
| TR | 0.25 |
| Q | 0.34 |

3



TRQ — 0.59
PS - 0.41.

4



No of characters in total = $\frac{(\text{\# of chars} \times \text{prob} \times 100)}{1}$.

| | #chars | |
|---|---|---|
| P | 2 | 44 |
| Q | 2 | 68 |
| R | 3 | 51 |
| S | 2 | 38 |
| T | 3 | 24 |

As there are 100 characters in total.

Total = 225 – option A.

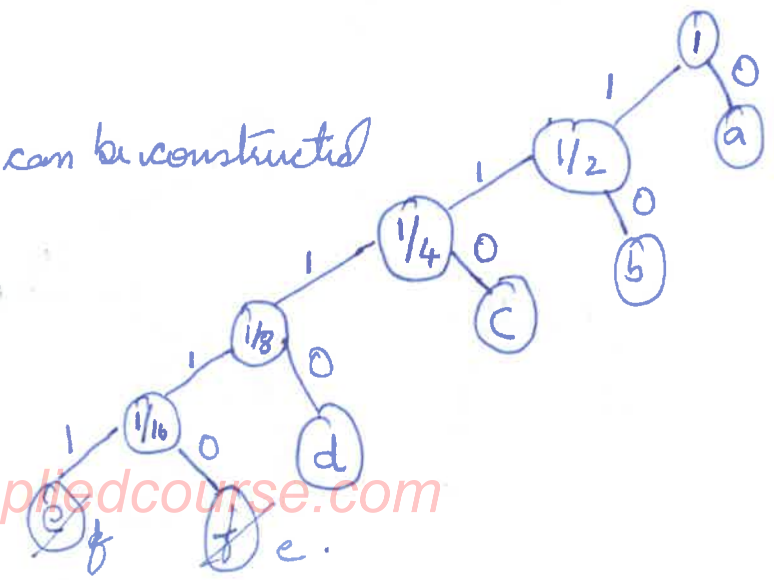
---

## 66.5 Solved Problem GATE 2007

Suppose the letters a, b, c, d, e, f have probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 respectively. Which of the following is the Huffman code for the letters a, b, c, d, e, f?

A. 0, 10, 110, 1110, 11110, 11111.
B. 11, 10, 011, 010, 001, 000
C. 11, 10, 01, 001, 0001, 0000.
D. 110, 110, 010, 000, 001, 111.

The huffman tree can be constructed as shown.



~~but there is no matching stg for~~

Option A seems to be matching but there is no matching for e and f so we can swap e and f as they are the two least probable characters.

Now option A is the matching option

---

## 66.6. Solved Problem GATE 2006.

Q) The characters a through h have the set of frequencies based on the first 8 Fibonacci Numbers as follows

a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21.

A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code?

110111100111010

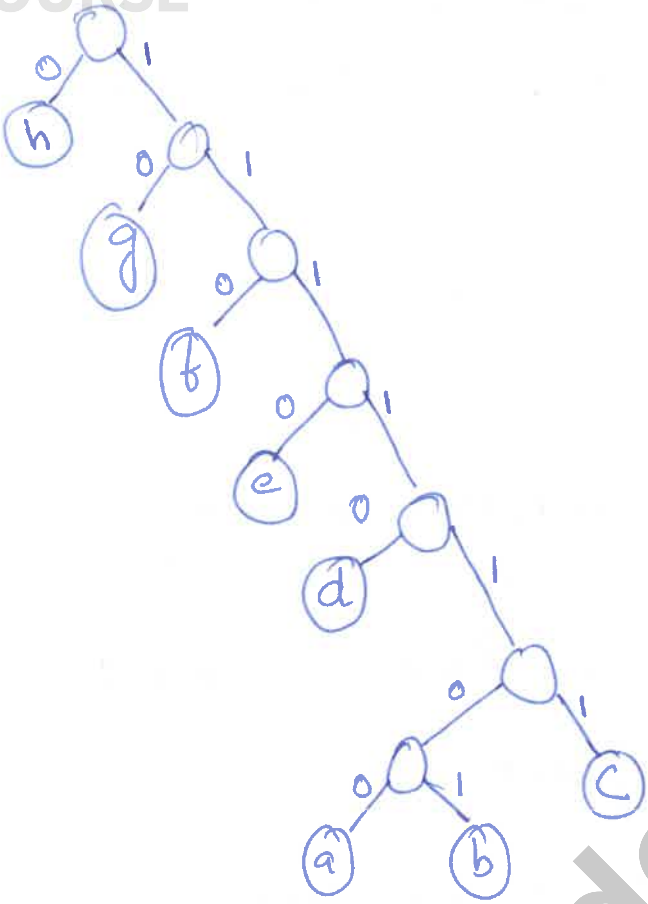
A. fdheg

B. ecgdf.

C. dchfg.

D. fehdg.

Solution :- The Huffman tree can be constructed by using the frequencies of the characters




110 1111001110101 = $\underbrace{110}_{f}$ $\underbrace{11110}_{d}$ $\underbrace{0}_{h}$ $\underbrace{1110}_{e}$ $\underbrace{10}_{g}$

= fdheg. option A.

## 66.7 Job Sequencing With Deadlines

- Given a set of jobs, each having associated profit and deadline associated with it.
- Every job takes 1 unit of time to complete
- Objective is to maximize the profit
- Constraint :- Respect the deadline.

| Job | Profit | deadline |
|---|---|---|
| $J_1$ | 85 | 5 |
| $J_2$ | 25 | 4 |
| $J_3$ | 16 | 3 |
| $J_4$ | 40 | 3 |
| $J_5$ | 55 | 4 |
| $J_6$ | 19 | 5 |
| $J_7$ | 92 | 2 |
| $J_8$ | 80 | 3 |
| $J_9$ | 15 | 7 |

→ First we have to be greedy about the profit, so we pick up the job with the maximum or most profit and place it ~~or~~ or schedule it as late as possible.

→ A Gantt Chart is a representation which is used to show which job is scheduled at what time slot.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $J_4$ | $J_7$ | $J_8$ | $J_5$ | $J_1$ | | $J_9$ |

1) $J_7$ has maximum profit 92 it is placed in 2nd slot

2) $J_1$ has max profit now is placed in 5th slot

3) $J_8$ has max profit now (80) it is placed in the 3rd slot.

4) $J_5$ has max profit now and is placed in the 4th slot.

5) $J_4$ has max profit now and is placed in 3rd slot but the 3rd slot is allocated to $J_8$ already and a slot $\leq 3$ is checked for

 2nd slot is also occupied now we check for the 1st slot it is free 
so it is allocated 1st slot.

6) Now $J_2$ is ~~allocated~~ is the one with max profit it is allocated 4, but it is already assigned to other job $J_5$ then we try to allocate a ~~job~~ slot, which is less than 4, but time slots 3, 2, 1 are already allocated to other more profitable jobs, so $J_2$ is not scheduled.

7) Next is $J_6$ (19) its deadline is 5 but all slots <=5 are already allocated and thus it is not scheduled.

8) Now next most profitable job is $J_3$ (16), its deadline is 3 but it also cannot be allocated any time slot because 3, 2, 1 are allocated to more profitable jobs.

9). Next most profitable job is $J_9$ its deadline is 7, it is allocated slot 7.

~~9) Next we are left with only one job that is.~~

Total profit by following this schedule = Profit($J_4$) + Profit($J_7$) + Profit($J_8$) + Profit($J_5$) + Profit($J_1$) + Profit($J_9$)

$= 40 + 92 + 80 + 55 + 85 + 15$

$= \underline{367}$.

— This is the maximum profit possible for these set of jobs and given deadlines.

— We are following greedy approach because we are choosing the most profitable job at every step

Ph: 844-844-0102

## Time Complexity

1. Initially we sort the jobs in $O(n \log n)$
2. Inserting the job in the Gantt Chart would require at most n comparisions and ~~inseting~~ inserting n jobs would require $O(n^2)$ time in the worst case

$\therefore$ Total time complexity $= O(n\log n + n^2) = O(n^2)$

## Space Complexity

- Additional $O(n)$ space is required to design the Gantt Chart.

$\therefore$ Space Complexity $\underline{O(n)}$

---

## 66.8 GATE 2005 Solved Problem.

- We are given 9 tasks $T_1, T_2, T_3, \ldots T_9$. The execution of each task requires one unit of time. We can execute one task at a time. Each task $T_i$ has a profit $P_i$ and a deadline $d_i$. The profit $P_i$ is earned if the task is completed before the end of the $d_i^{th}$ unit of time.

| Task. | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ |
|---|---|---|---|---|---|---|---|---|---|
| Profit | 15 | 20 | 30 | 18 | 18 | 10 | 23 | 16 | 25 |
| Deadline | 7 | 2 | 5 | 3 | 4 | 5 | 2 | 7 | 3 |

Are all the tasks completed in the schedule that gives maximum profit?

A. All tasks are completed.

B. $T_1$ and $T_6$ are left out.

C. $T_1$ and $T_8$ are left out.

D. $T_4$ and $T_6$ are left out.

Mail: Gatecse@appliedcourse.com

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $T_2$ | $T_7$ | $T_9$ | $T_5$ | $T_3$ | $T_1$ | $T_8$ |

1. $T_3$ is allocated 5
2. $T_9$ is allocated 3
3. $T_7$ is allocated 2 → 4. $T_2$ is allocated 2, but already occupied its allocated 1.
5. Either $T_4$ or $T_5$ can be selected as we have $T_4$ in the options lets select $T_5$
   $T_5$ is allocated 4
6. $T_4$ is allocated 3 but 3, 2, 1 are already occupied, ∴ $T_4$ is not scheduled.
7. $T_8$ is allocated 7
8. $T_1$ is allocated 7 but it is already occupied, it is allocated 6.
9. $T_6$ is allocated 5. but 5, 4, 3, 2, 1 are already allocated, ∴ $T_6$ is not scheduled.

∴ $T_4$ and $T_6$ are left out option D is correct.

---

## 66.9 Optimal Merge Pattern

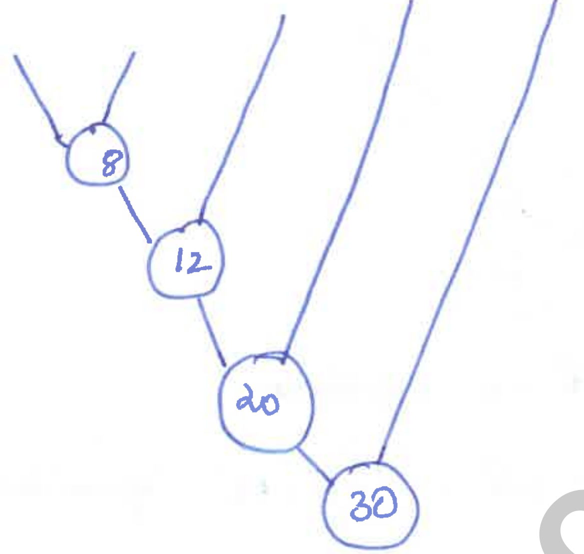
- As in the case of merge sort we merge 2 lists which are sorted into 1 list., there could also be n lists which we are to merge they could be merged using either n way merge or in which all the n lists are merged at once into a list or we could merge two two lists at a time in other words we could perform (n-1) 2 way merges to merge into one single list.

- The time complexity of merging two lists which are of size or length m and n elements is given by $O(m+n)$.

Let us take the following example.

$n = 5$ lists

| lists | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ |
|---|---|---|---|---|---|
| length | 6 | 2 | 4 | 8 | 10. |




first we combine $L_1$ & $L_2$ It takes $6+2=8$ comparisons

then $(L_1+L_2)$ and $L_3$ It takes $8+4=12$ comparisions

then $(L_1+L_2+L_3)$ and $L_4$ It takes $12+8=20$ comparisions.

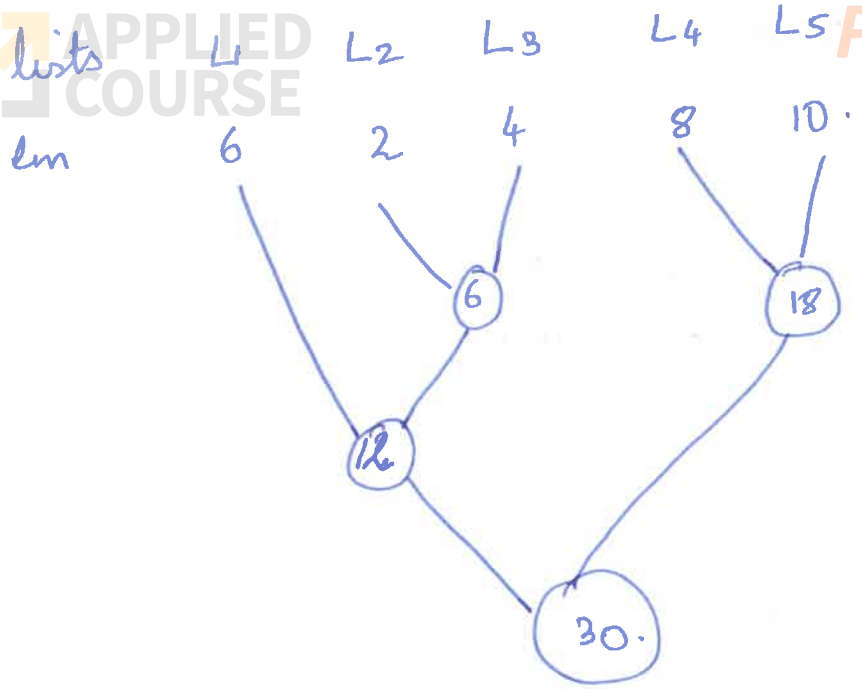
then $L_1+L_2+L_3+L_4$ and $L_5$ It takes $20+10=30$ comparisions

Total no of comparision operations = 70 operations.

Can we merge the lists in of by fewer operations?

We have to identify the optimal merge pattern → Minimal cost of $(n-1)$ 2 way merges.

Greedy Approach :- Always pick the 2 smallest lists to merge at every step.




1. $L_2$ & $L_3$ are merged $= 6$ operations

2. $L_2 + L_3$ and $L_1$ are merged $= 6 + 6 = 12$ operation

3. $L_4$ and $L_5$ are merged $= 8 + 10 = 18$ operations.

4. $L_1 + L_2 + L_3$ and $L_4 + L_5$ are merged $= 12 + 18 = 30$ operations

Total # operations $= 6 + 12 + 18 + 30$

$= \underline{66}$.

- The optimal way to solve this problem is by using the greedy approach.

## Time Complexity

- We are performing $(n-1)$ merges.

∴ - In each merge we are merging 2 least/smallest lists

- list length we can maintain a min heap

At each step 1. we extract min twice — $2 \times \log n$ } $O(\log n)$

2. insert their sum once — $\log n$

- Total cost $(n-1) \times \log(n) = O(n \log n)$

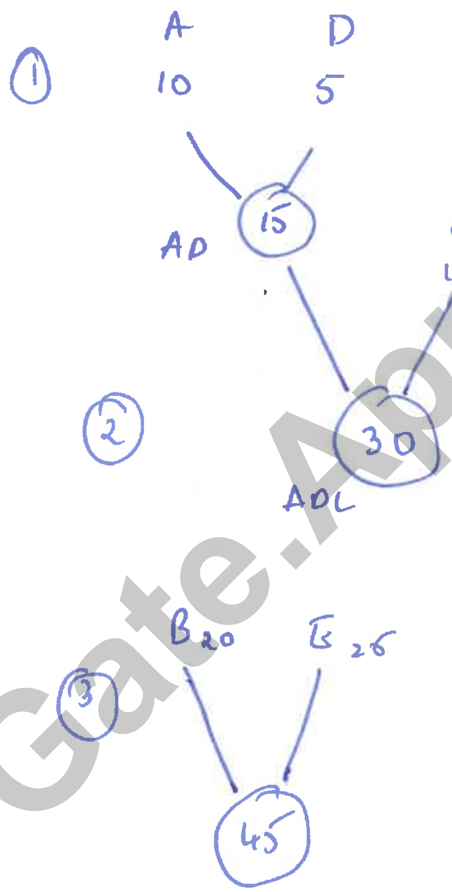
# 66.10 Solved Problem : GATE 1999

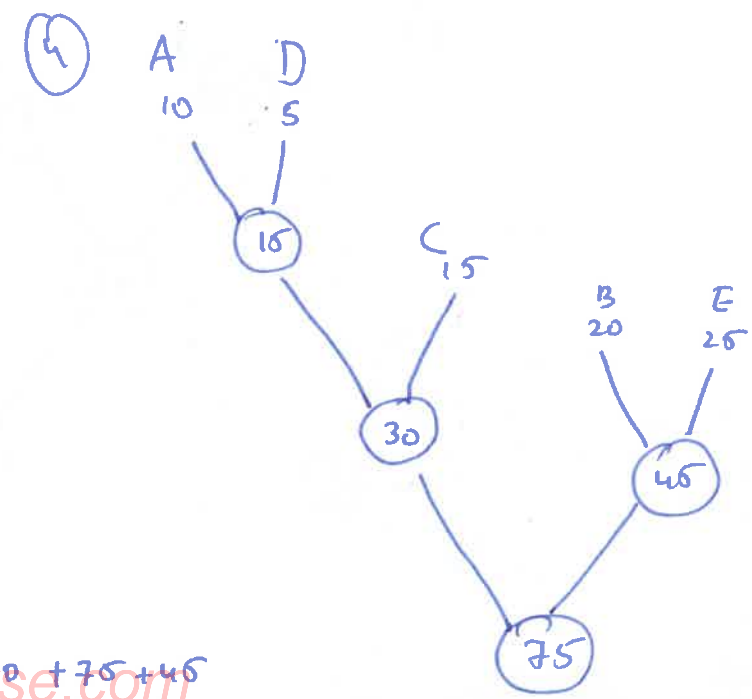


- The minimum number of record movements required to merge five files A (10 records), B (15 records), D (5 records) and E (25 records) is ___.

A. 165
B. 90
C. 75
D. 65

| A | B | C | D | E |
|---|---|---|---|---|
| 10 | 20 | 15 | 5 | 25 |



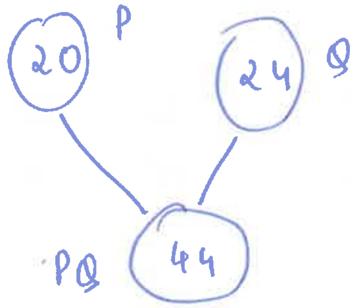

Cost or No of moves = 15 + 30 + 75 + 45
= 165 A option.

# 66.11 Solved Problem GATE 2014

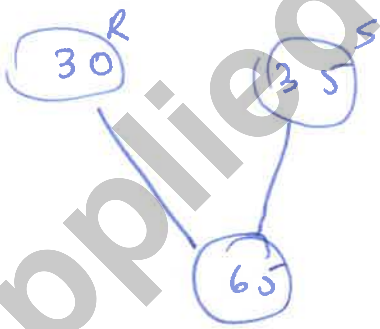


Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is ____.

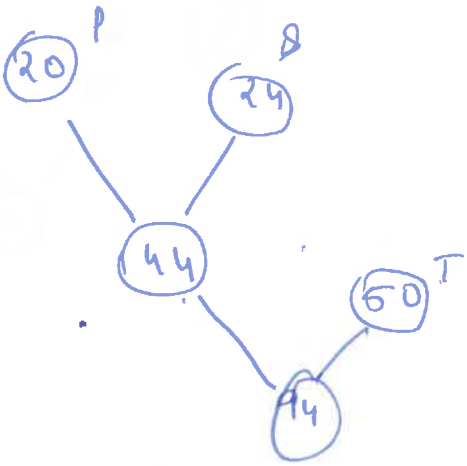
1)




2)

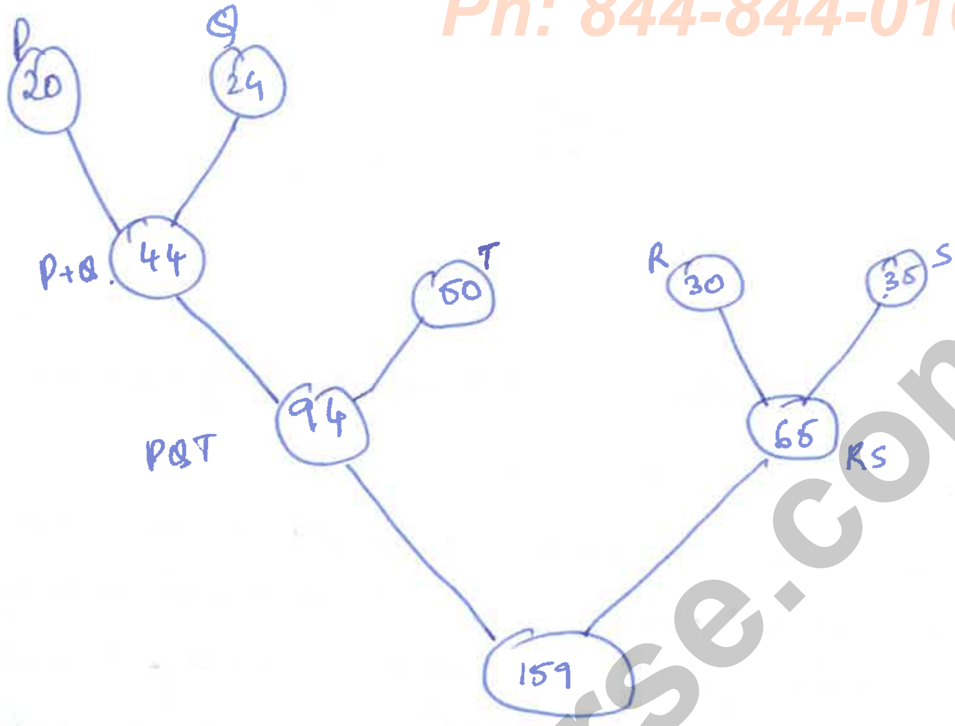



3)




4)

4)




~~# of comparisions = 44 + 94 + 65 + 159~~.

# of comparisions in merging two lists of m and n = $m+n-1$

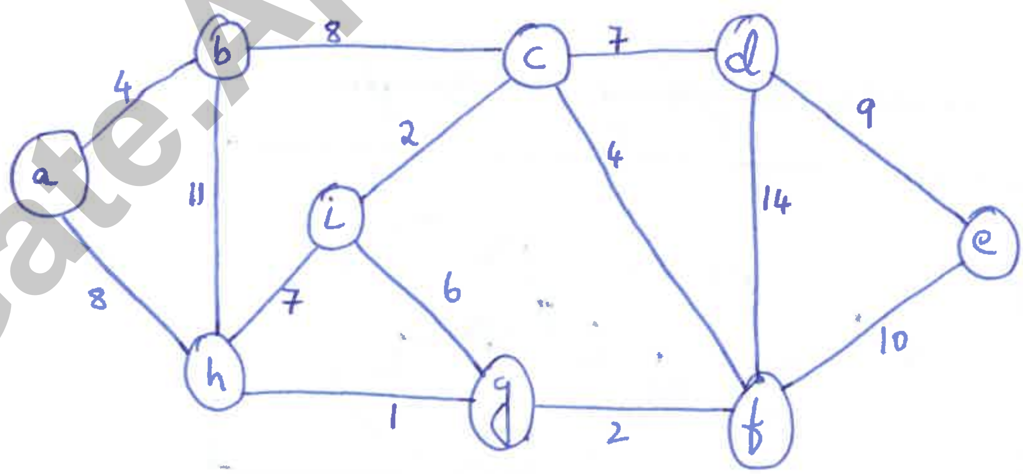
∴ # of comparisions = $43+93+64+158$

$= 358$

---

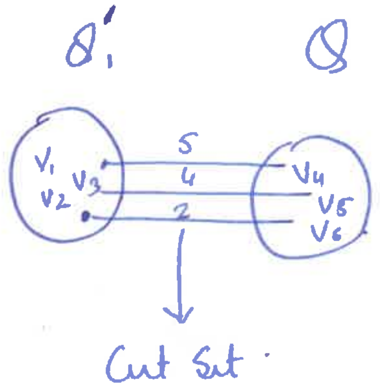
## 66.12 Minimum Spanning Trees

### Prim's Algoritm.




Start :- We start with an empty spanning tree i.e. with no edges but only vertices.

Two sets :- We try to maintain two sets one is the set of vertices for which we have already computed the MST (Q') and other is for which we have to compute (Q)



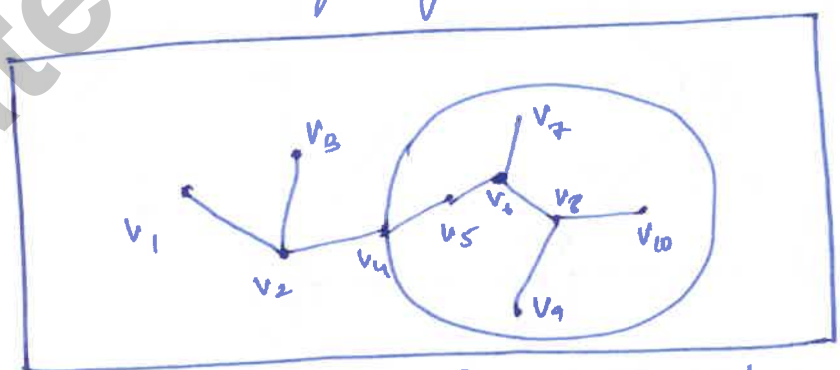

→ In each iteration we determine the set of edges which connect from $S'$ to $S$ this is known as the cut set.

→ And then from the cut set we pick the edge which is the most cheapest or least cost, and include it in our minimal spanning tree. The vertex which is a part of $S$ and at the other end of the cut vertex is added to $S'$ now. In the above example vertex edge with wt 2 is added and $V_6$ is added to set $S'$ and removed from $S$.

— Hence the greedy choice at every step is that we are choosing the minimal cost cut vertex at every step.

— The Optimal Substructure property :- Subtree's of MST's are also MST's.

Let us consider the following MST example

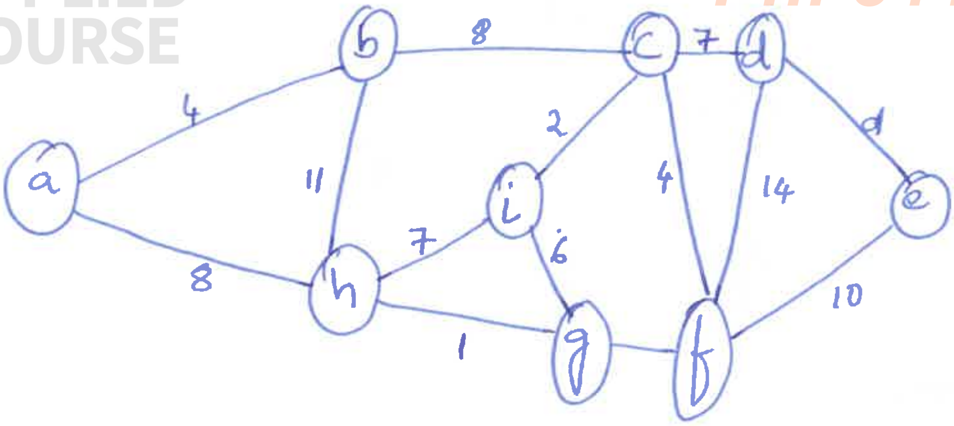



If we consider the sub tree of the MST which is circled, the subtree is also an MST which is the MST of the subgraph of the original graph with these vertices. i.e $V_4, V_5, V_6, V_7, V_8, V_9, V_{10}$.

Let us compute the MST for the example graph shown.







Initially $Q' = \{\}$

$Q = \{a, b, c, d, e, f, g, h, i\}$

We add the vertex a initially a in set $Q'$.

$Q' = \{a\}$

$Q = \{b, c, d, e, f, g, h, i\}$

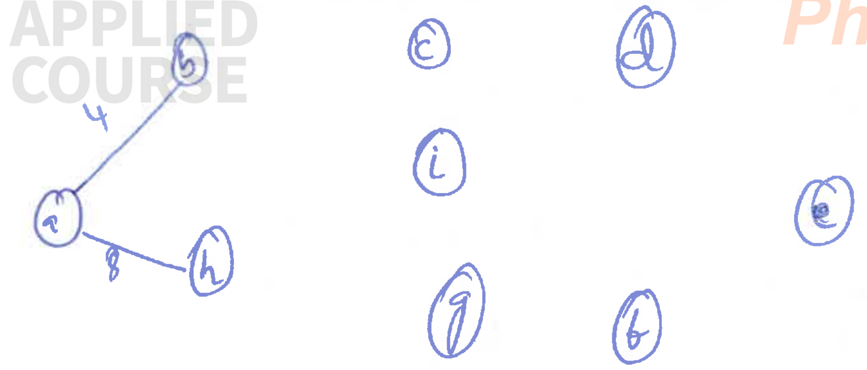
The cut set = $\{4, 8\}$

Now 4 is added to MST

$Q' = \{a, b\}$

$Q = \{c, d, e, f, g, h, i\}$.

Cut set $\{\underset{ah}{8}, 11, \underset{bc}{8}\}$



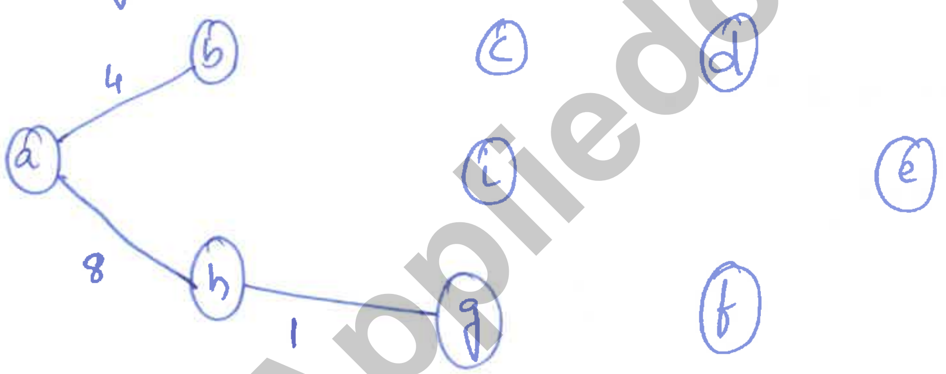
adding 8 (ah).

$Q' = \{a, b, h\}$

$Q = \{c, d, e, f, g, i\}$

Cut Set = $\{8, 7, 1\}$

choosing 1

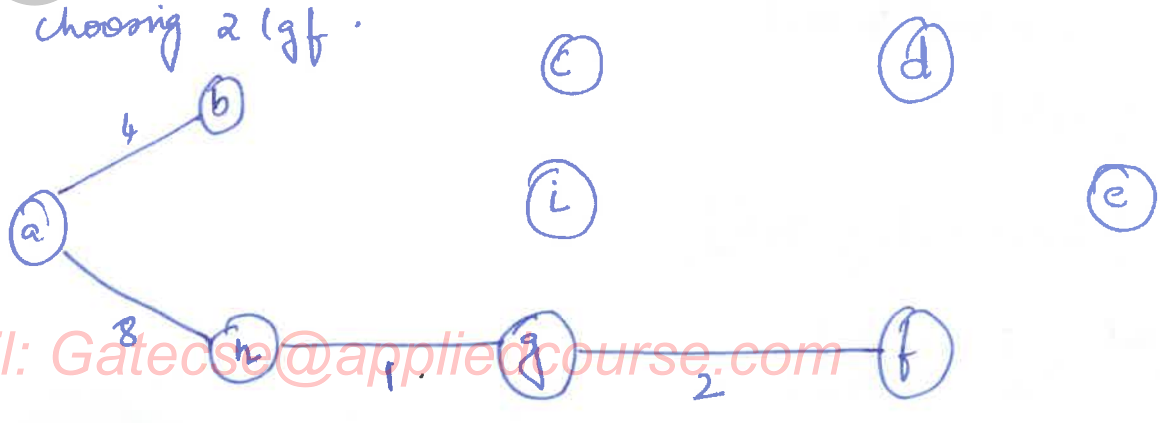


$Q' = \{a, b, h, g\}$

$Q' = \{c, d, e, f, i\}$.

Cut Set = $\{\underset{gf}{2}, 6, 7, \underset{bc}{8}\}$

choosing 2 (gf.

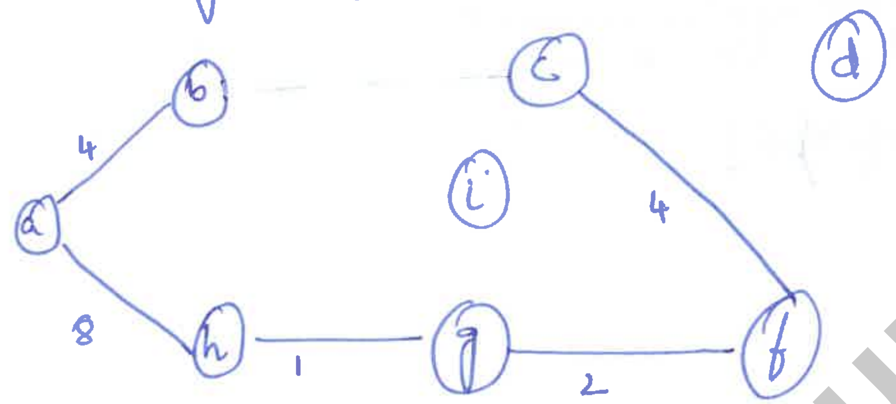


$S' = \{a, b, h, g, f\}$

$S = \{c, d, e, i\}$

Cut Set $\{6, 7, \underset{cf}{4}, \underset{fd}{4}, \underset{fe}{10}, \underset{bc}{8}\}$

choosing 4 (cf)



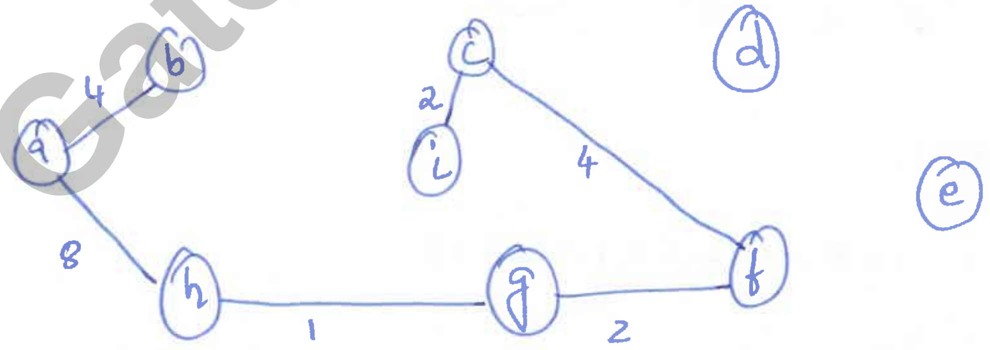
Now

$S' = \{a, b, c, h, g, f\}$

$S = \{d, e, i\}$

Cut Set $= \{10, 9, \underset{cd}{7}, 2, 6, \underset{hi}{7}\}$

Adding 2 ci



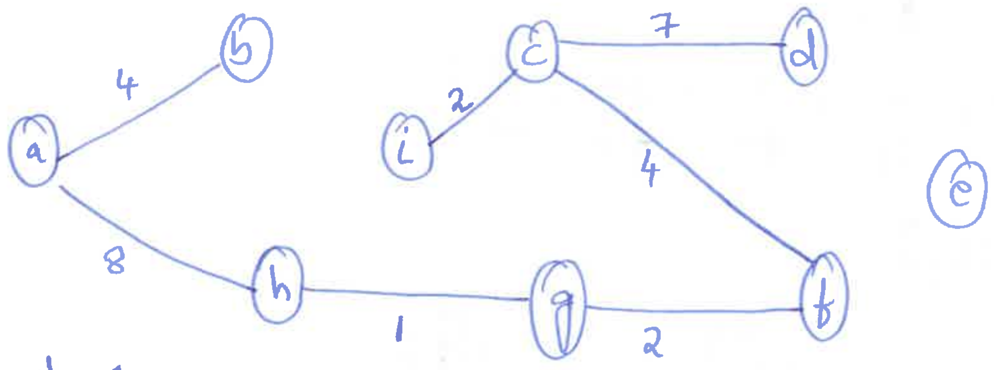
Now

$S' = \{a, b, c, f, g, h, i\}$

$S = \{d, e\}$ & Cut Set $= \{7, 10, 14\}$

Now choosing 7 c,d






$S' = \{a, b, c, d, f, g, h, i\}$

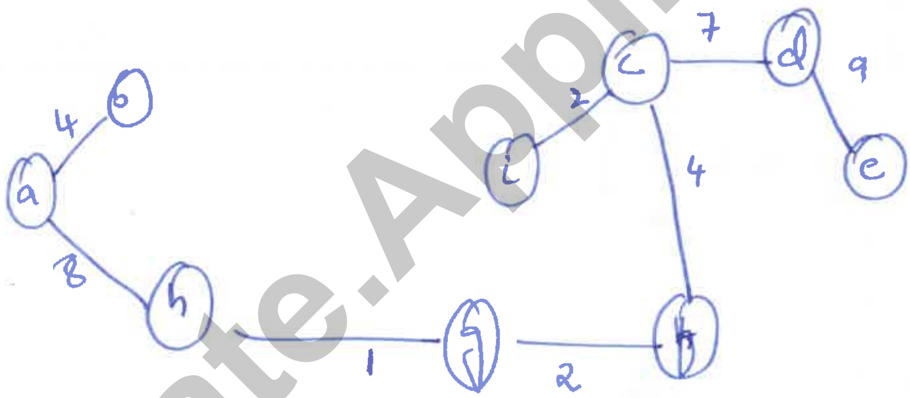
$Q = \{e\}$

Cut set $= \{9, 10\}$

choosing 9 de:

$S' = \{a, b, c, d, e, f, g, h, i\}$

$Q = \{\}$.




The above is the MST.

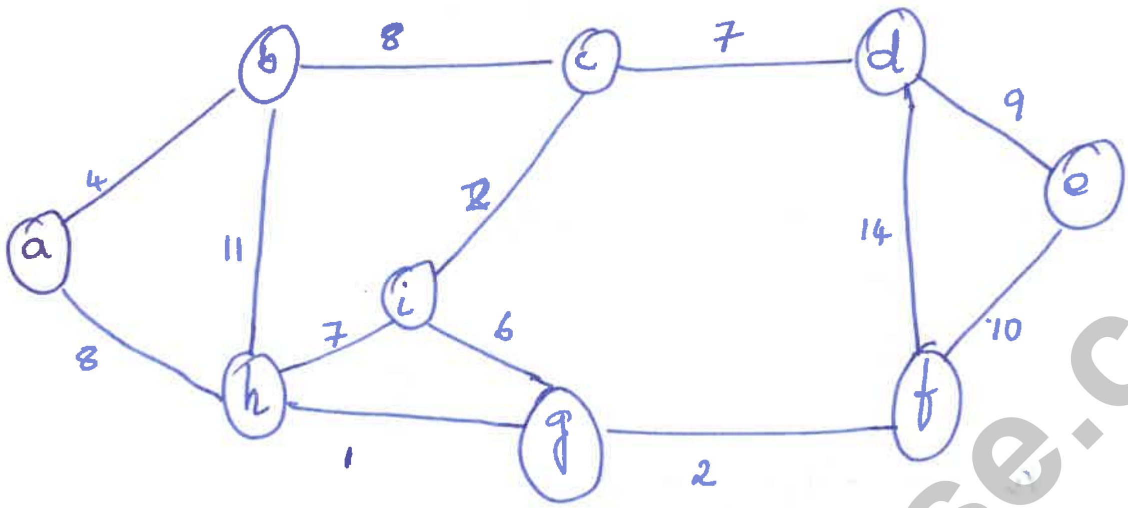
Cost of the MST $= 4+8+1+2+4+2+7+9$

$= 37$

# 66.13. Minimum Spanning Trees



## Greedy Krushkals' Algorithm




→ Each vertex is placed into a separate set. in all initally we will have no of sets equal to the number of vertices.

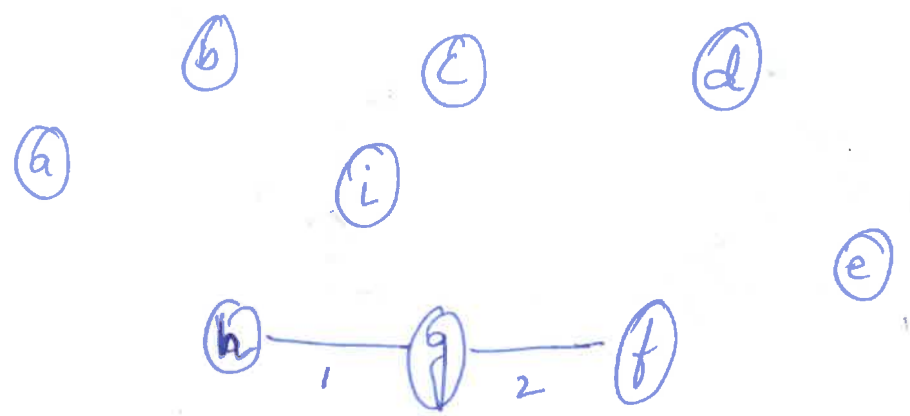
→ At each step we pick up the edge with the minimum / least cost and if it is connecting vertices of two diffent sets then it is added to the MST, if it is connecting two vertices of the same set then such a edge is discarded and not added to the MST.

In the above example graph.

1. {a} {b} {c} {d} {e} {f} {g} {h} {i}

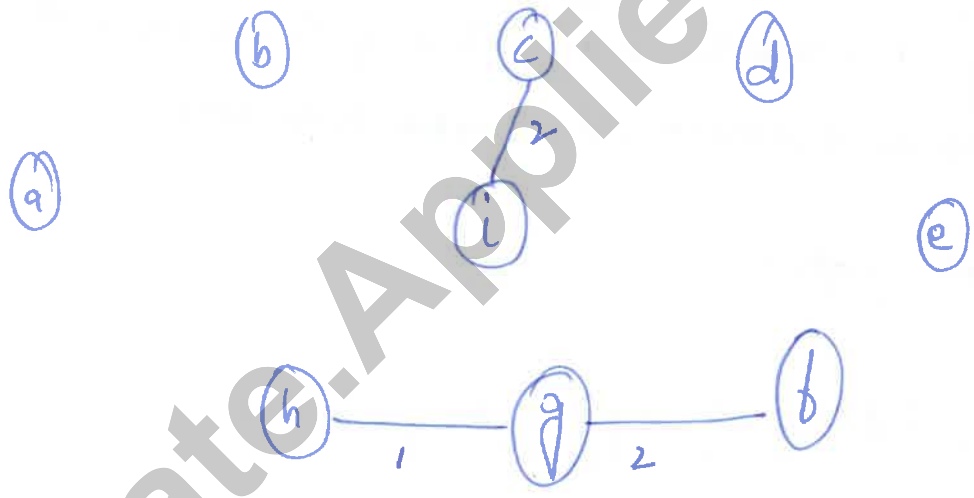
2. edge (h-g) cost 1 ~~{h}~~ h, and g belong to different sets. combining the two edges into one set.

Now we have edge g-f of cost 2 and edge c-i of cost 2
any one can be choosen, lets choose g f.



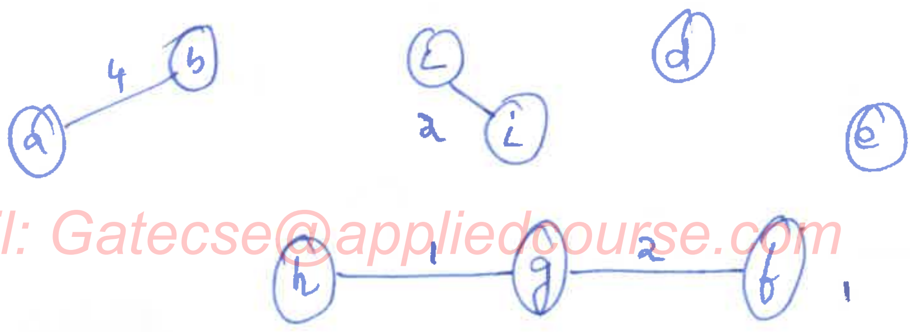

The sets are $\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f,g,h\}$ $\{i\}$.

Now choosing c-i



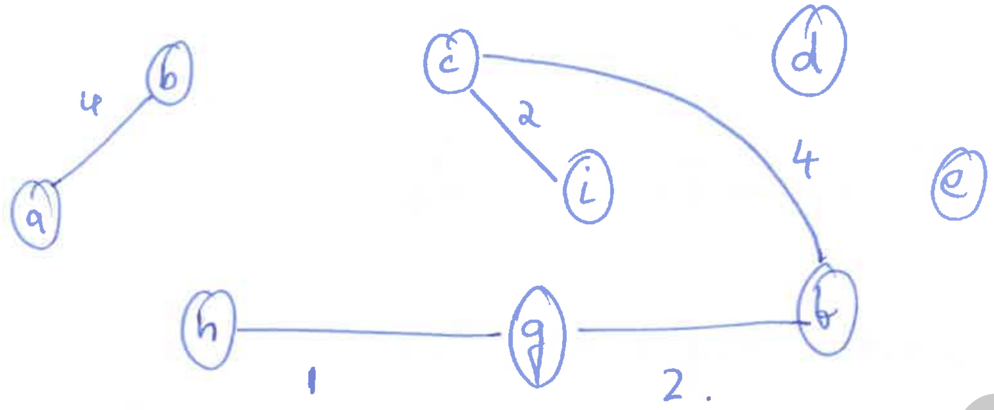

The sets are $\{a\}$ $\{b\}$ $\{c,i\}$ $\{d\}$ $\{e\}$ $\{f,g,h\}$

Now next edge which is cheapest = a-b cost = 4.



The sets are {a, b} {c, i}, {d} {e} {f, g, h}

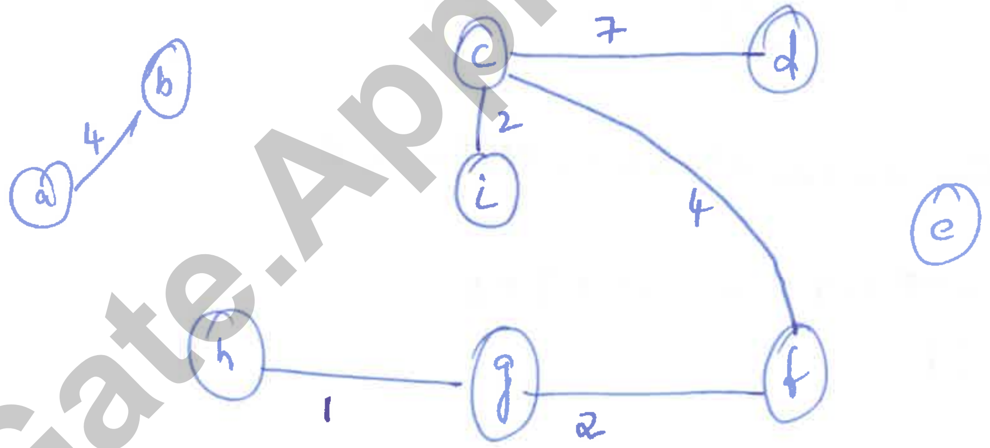
The next cheapest edge is c-f cost = 4 it is connecting different sets




The sets are {a, b} {c, f, g, h, i} {d} {e}

→ The next edge is g-i cost = 6, but this edge connects vertices of the same set so we need to discard this edge.

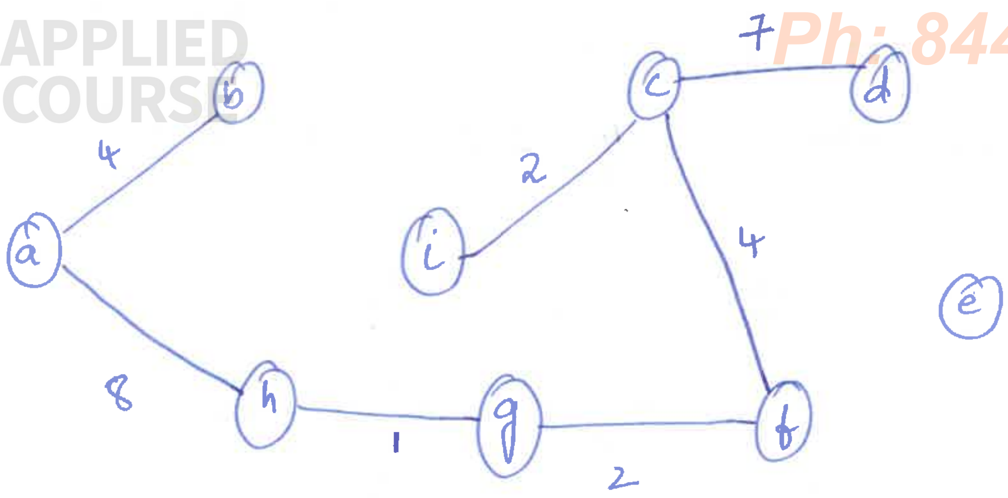
→ Next edge is c-d cost = 7




The sets are {a, b} {c, f, g, h, i, d} {e}

→ h-i is an edge of cost 7 but it connects two vertices of the same set.

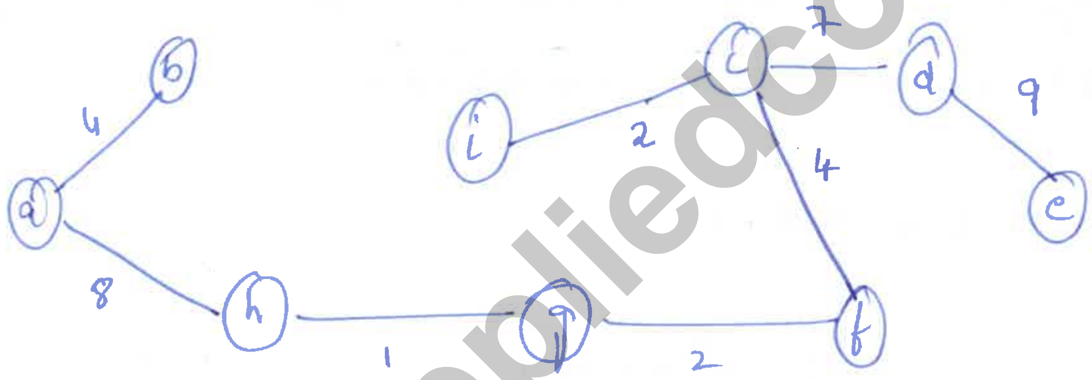
→ a-h is the next edge of cost = 8




The sets are . {a, b, c, d, f, g, h, i} {e}

Now the next edge is b-c cost 8 but it connects vertices of the same set so we need to discard it.

Next we take the edge de cost - 9 .





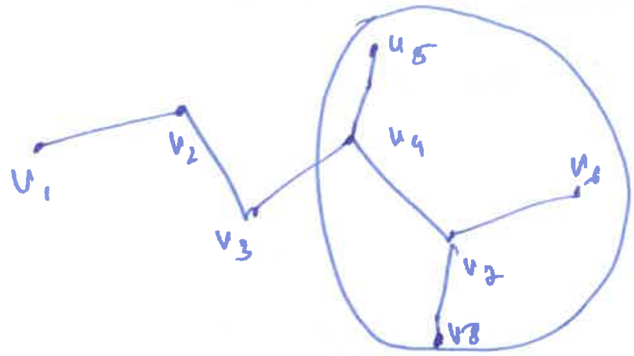

Now all the vertices are covered and the MST is full.

Cost of the MST = 4+8+1+2 + 4+2+7+9
= 37

Note* :- For a given graph there can be multiple Minimal spanning trees but the cost of the MST will always be the same.

- At each step we are choosing the least cost edge greedily which is the reason why it satisfies the greedy property

It also satisfies the optimal substructure property, as the subtree of the MST is also an MST.




→ In the above MST if we consider a subtree of it then the subtree (circled) will be MST of the graph which is the subgraph of the original graph. (containing vertices $v_5, v_4, v_7, v_6, v_8$).

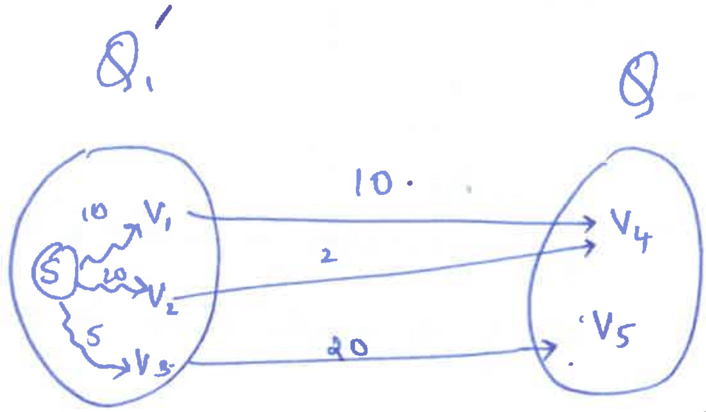
## 66.14 GREEDY ALGORITHM : DIJKSTRA'S ALGORITHM

→ Single Source shortest path problem :- Given a graph and a source vertex in that graph the objective is to find the shortest path from the source to each of the other vertices in the graph.

→ The Dijkstra's algorithm is guaranteed to work properly if there are no -ve weight edges.

→ As in the prims algorithm we are maintaing 2 sets $Q'$ is the set of vertices for which the shortest path is known and $Q$ is the set of vertices for which the shortest path from the source vertex is to be calculated.

→ At each step we have to follow or find the cost to vertices in Q from Q' and take/choose the one with minimal cost /greedyly.
The choosen vertex is added to the set Q'.



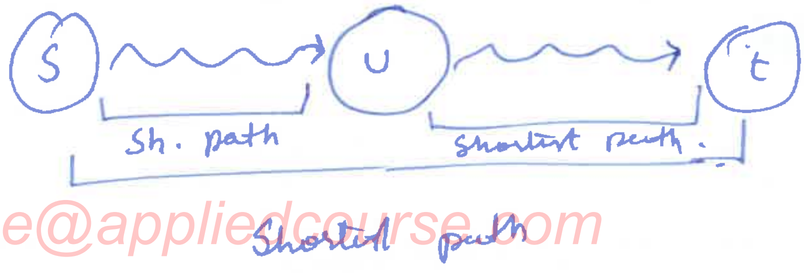

In the above example if we use the edge $V_1$ to $V_4$ we get a path of cost 20

| | | | | |
|---|---|---|---|---|
| " | | $V_2$ to $V_4$ | " | 22 |
| " | | $V_3$ to $V_5$ | " | 25 |

The minimum is 20 we add $V_4$ to our set Q' and choose it.

— Here we are greedy in the cost of the path at each step we are choosing the ~~edg~~ path which is having the least cost.
The greedy property is satisfied here.

— The optimal Substructure property for shortest distance path is if s to t is the shortest path from via vertex u then the sub path s to u is also the shortest path from S to U.





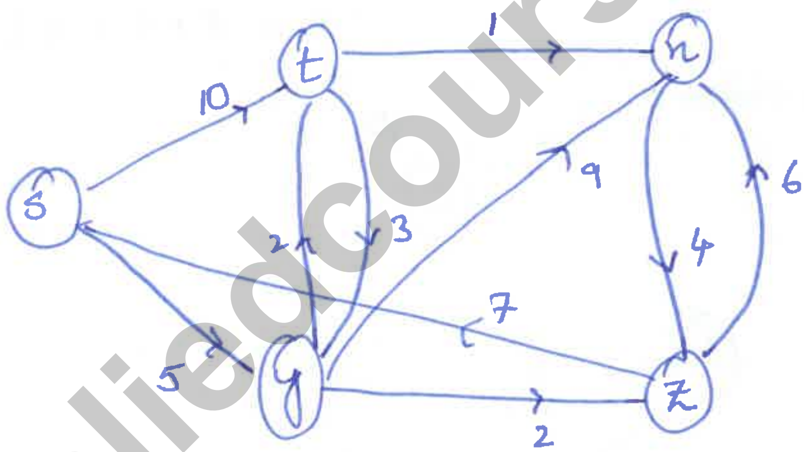



—

We can proove this property by countradiction easily.

- let us assume that there exist a path shorter than s to u
- Then the shortest path from s to t must contain the shorter path, otherwise the path s to t is not the shortest path, hence there does not exist a shorter subpath and s to t is the shortest subpath from s to t,

Let us apply Dijkstra's algorithm on the example graph shown below.




Initally $S' = \phi$
$S = \{s, t, x, y, z\}$

Now we add the source to $S'$

$S' = \{s\}$

$S = \{t, x, y, z\}$

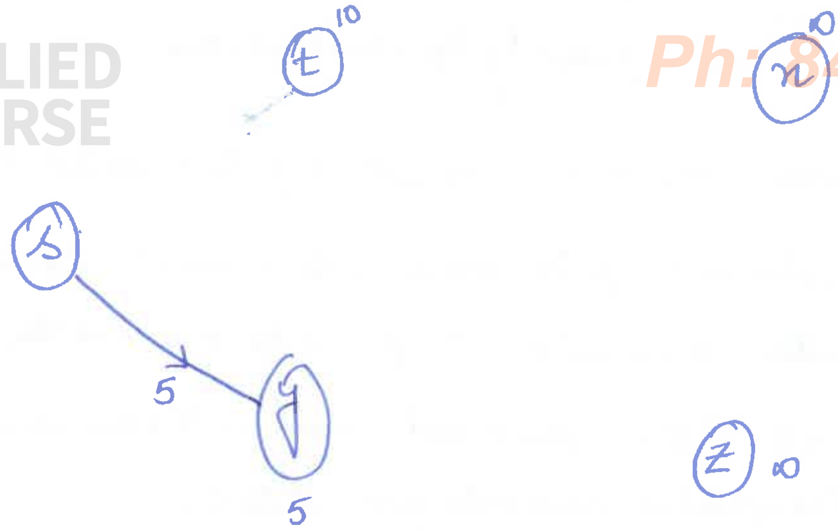
cost (Initially all except the source are set to $\infty$)
$s = 0$
$t = \infty$
$x = \infty$
$y = \infty$
$z = \infty$.

From $S'$ we can reach $t = 0 + 10 = 10$
$y = 0 + 5 = 5$

we choose y.

Now $S' = \{s, y\}$
$S = \{t, x, z\}$

cost
$s = 0$ $x = \infty$
$t = 10$ $z = \infty$.
$y = 5$

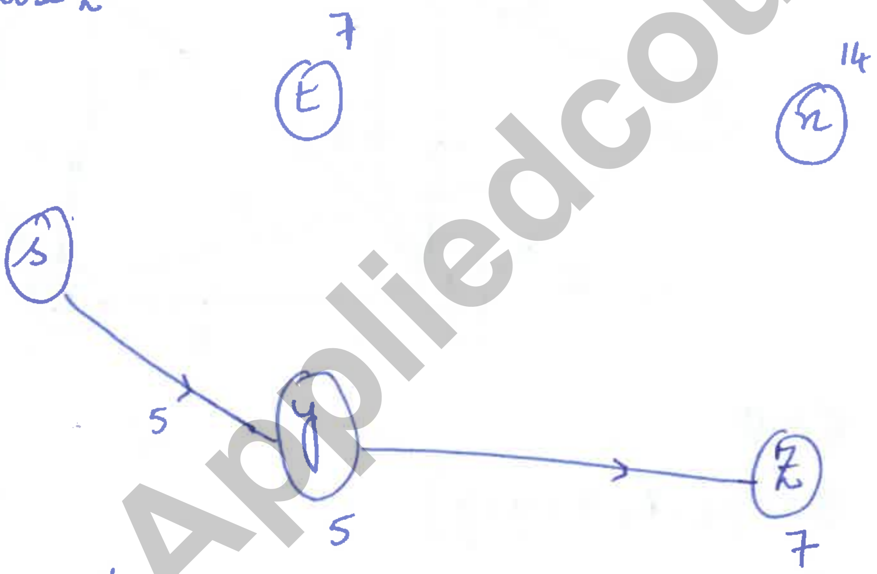



Now From S' we can reach $z = 5 + 2 = 7$

$t = 5 + 2 = 7$ ($< 10$ previous cost)

$x = 5 + 9 = 14$ ($< \infty$ previous cost)

Lets choose z



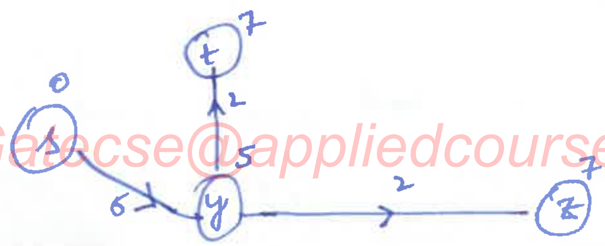

Now $S' = \{s, y, z\}$

$S = \{t, x\}$

Now from S' we can reach x

$x = 7 + 6 = 13$ ($< 14$ previous cost)

The least is of t (cost = 7) Adding t to S'



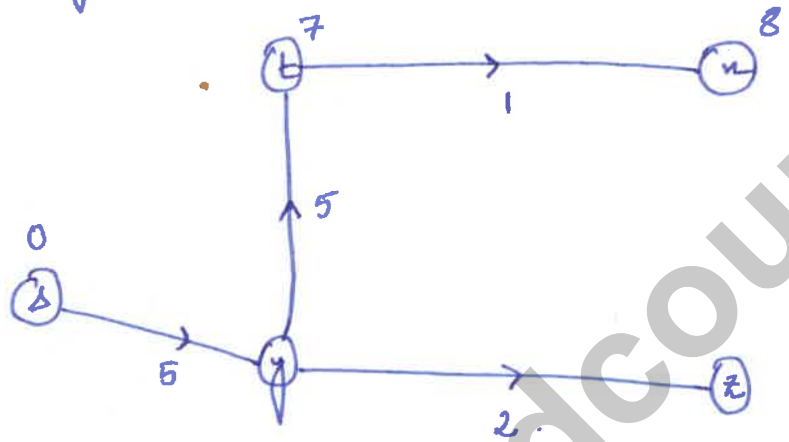
Now $S' = \{s, t, y, z\}$

$S = \{x\}$

from $S'$ we have a new path to $x$

$x = 7 + 1 = 8$ ($< 13$ previous cost)

Adding $x$ to $S'$




Now $= S' = \{s, t, x, y, z\}$

$S = \phi$

Now we have reached the end and all the vertices have been explored.